Tugas Akhir – KI141502

RANCANG BANGUN MODUL PEMBANGKIT PETA DAN LEVEL DINAMIS UNTUK *GAME* PANDAWA: THE HEROES OF NGASTINA PADA PLATFORM FLASH

RIZKA NOVIANA INDRIYANI
NRP 5110 100 133

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman Hariadi, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

FINAL PROJECT – KI141502

DESIGN AND IMPLEMENTATION OF DYNAMIC MAP AND LEVEL GENERATOR MODULE FOR PANDAWA: THE HEROES OF NGASTINA GAME ON THE FLASH PLATFORM

RIZKA NOVIANA INDRIYANI
NRP 5110 100 133

Advisor
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman Hariadi, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

LEMBAR PENGESAHAN

RANCANG BANGUN MODUL PEMBANGKIT PETA DAN LEVEL DINAMIS UNTUK *GAME* PANDAWA: THE HEROES OF NGASTINA PADA *PLATFORM* FLASH

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Interaksi, Grafika dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
RIZKA NOVIANA INDRIYANI
NRP. 5110 100 133

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Imam Kuswardayan, S.Kom., M.T. .....
NIP: 197612152003121001 (pembimbing 1)

Ridho Rahman Hariadi, S.Kom, M.Sc. .....
NIP: 198702132014041001 (pembimbing 2)

SURABAYA
JUNI, 2015

RANCANG BANGUN MODUL PEMBANGKIT PETA DAN LEVEL DINAMIS UNTUK *GAME* PANDAWA: THE HEROES OF NGASTINA PADA *PLATFORM* FLASH

Nama Mahasiswa : Rizka Noviana Indriyani
NRP : 5110 100 133
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing I : Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing II : Ridho Rahman Hariadi, S.Kom., M.Sc.

Abstrak

Dalam game strategi perang, peta merupakan hal yang sangat penting. Peta digunakan sebagai area pertempuran dalam game strategi perang. Peta tersebut biasanya terdiri dari terrain, laut, bangunan, hutan, obstacle, dan lain-lain. Dalam beberapa game strategi perang, peta dan level dibentuk secara dinamis menggunakan teknik Procedural Content Generation. Bagi pemain, hal ini akan membuat game semakin menantang dan menambah experience pemain. Pembangkitan peta dan level dinamis juga akan membantu dari sisi pengembangan game karena dapat mengurangi biaya pengembangan dan mempermudah pembuatan peta dan level secara otomatis.

Dalam Tugas Akhir ini, akan dikembangkan Modul Pembangkit Peta dan Level Dinamis yang akan menangani pembangkitan peta dan kombinasi musuh untuk game Pandawa: the Heroes of Ngastina pada platform flash. Dalam game ini, bentuk peta tidak mempengaruhi tingkat kesulitan level. faktor yang mempengaruhi tingkat kesulitan level adalah kombinasi musuh yang dihasilkan. Peta dinamis dibangun dengan menggunakan metode Cellular Automata. Kombinasi musuh dipengaruhi oleh skor pemain yang akan diolah menjadi rating dengan metode Elo Rating System. Rating tersebut akan menjadi nilai modal untuk membangkitkan kombinasi musuh pada level selanjutnya dengan mengadaptasi Coin-Changing Problem menggunakan algoritma Dynamic Programming. Dari hasil pembangkitan peta, akan didapatkan

peta yang berubah-ubah tiap level sesuai dengan rasio perbandingan area dan rule pembentukan peta yang didefinisikan pada game. Sedangkan dari pembangkitan musuh, akan didapatkan kombinasi musuh yang menyesuaikan kemampuan pemain dengan jumlah musuh maksimal lima orang pada tiap level.

Kata kunci: game strategi, game perang, pembangkit level dinamis, pembangkit peta dinamis.

DESIGN AND IMPLEMENTATION OF DYNAMIC LEVEL AND MAP GENERATOR MODULE FOR PANDAWA: THE HEROES OF NGASTINA GAME ON THE FLASH PLATFORM

Student Name	: Rizka Noviana Indriyani
NRP	: 5110 100 133
Major	: Teknik Informatika FTIf-ITS
Advisor I	: Imam Kuswardayan, S.Kom., M.T.
Advisor II	: Ridho Rahman Hariadi, S.Kom. M.Sc.

Abstract

In a war strategy game, map is very important. Map is used as a battlefield in a war strategy game. These maps are usually consist of terrains, oceans, buildings, forests, obstacles, and others. In some war strategy games, maps and levels are dynamically generated using Procedural Content Generation techniques. For players, this will make the game more challenging and adds the player's experience. Generation of dynamic maps and levels will also help in game development because it can reduce development costs and help the making of maps and levels automatically.

This Final Project will develop Dynamic Map and Level Generator Module that will handle the generation of maps and combination of enemy for Pandawa: the Heroes of Ngastina game. In this game, the form of map did not affect the level of difficulty level. The difficulty of each level is affected by combination of enemy. Dynamic maps are built using Cellular Automata. Combination of enemy is determined by the player's score that will be processed into the rating using method Elo Rating System. The rating will be the changing value to generate a combination of enemies on the next level by adapting Coin-Changing Problem using Dynamic Programming algorithms. From the generation of

the map, we will get the map that varies each level in accordance with the ratio of the area and the establishment of the rule is defined on the game map. Whereas from the enemy generation, we will get the combination of enemies that adjust the ability of the player with the maximum number of enemy is five on each level.

Keywords: strategy game, war game, dynamic level generation, dynamic map generation.

KATA PENGANTAR

Segala puji dan syukur, ke hadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

"Rancang Bangun Modul Pembangkit Peta dan Level Dinamis pada *Game* Pandawa: the Heroes of Ngastina pada *Platform* Flash".

Melalui lembar ini, penulis ingin menyampaikan ucapan terima kasih dan penghormatan yang sebesar-besarnya kepada:

1. Allah SWT atas segala nikmat dan rahmat yang telah diberikan selama ini.
2. Kedua orang tua penulis, Bapak Khusaini dan Ibu Siyami, serta adik tercinta, Muhammad Arrizal Zamzami, dan semua keluarga yang telah mencurahkan kasih sayang, perhatian, dan doa kepada penulis selama ini.
3. Bapak Imam Kuswardayan dan Bapak Ridho Rahman Hariadi selaku dosen pembimbing yang telah memberikan bimbingan, motivasi, dan meluangkan waktu untuk membantu pengerjaan Tugas Akhir ini.
4. Bapak dan Ibu dosen Teknik Informatika ITS yang telah membina dan memberikan ilmu yang sangat bermanfaat kepada penulis selama menimba ilmu di Teknik Informatika ITS.
5. Maulidan Bagus Afridian Rasyid yang telah memberikan banyak ilmu, bantuan, dan *support* dalam banyak hal selama ini.
6. Rekan-rekan kerja di Maulidan Games Productions (Maya Aulianissa, Ika Nurul Fauziyah, Ramadhany Candra Arif Putra, Pramono Adhi Widagdo, Dirhamsyah Gunawan, Julian Akbar Renaldi, Reno Dwi Gustavino, dan Erlan

Kurnia) yang memberikan saya banyak ilmu baru dan membuat saya berkembang.

7. Ibu Dyah Sari Wardani dan Ibu Ratih, yang selalu memberi banyak dukungan dan nasehat kepada serta menjadi ‘keluarga’ ketika saya menempuh masa studi di Surabaya.
8. Rekan-rekan Teknik Informatika ITS terutama angkatan 2010 atas segala dorongan semangat dan ilmu yang diberikan.
9. Pihak-pihak lain yang tidak sempat penulis sebutkan, yang telah membantu kelancaran pengerjaan Tugas Akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun Tugas Akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, Juni 2015
Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bagian ini akan dijelaskan hal-hal yang menjadi latar belakang pembuatan Tugas Akhir, permasalahan yang dihadapi, batasan masalah, tujuan dan manfaat yang ingin dicapai dalam pengerjaan Tugas Akhir, metodologi pembuatan Tugas Akhir, dan sistematika penulisan yang digunakan dalam pembuatan Tugas Akhir ini.

1.1 Latar Belakang

Turn-Based Strategy Game adalah jenis *genre game* strategi dimana pemain mempunyai giliran untuk melakukan suatu aksi. Setelah giliran seorang pemain selesai, maka pemain tersebut harus menunggu giliran berikutnya untuk melakukan aksi lainnya [16]. Dalam *game Turn-Based Strategy* khususnya strategi perang, peta merupakan hal yang sangat penting. Sebuah peta dalam *game* strategi perang biasanya terdiri dari *terrain*, laut, bangunan, hutan, tambang emas, dan lain-lain. Beberapa *game* strategi perang menggunakan *fog-of-war* sehingga ada sebagian wilayah yang masih tertutup (*covered*). Wilayah tersebut akan terbuka (*visible*) jika pemain sudah menjelajahi daerah tersebut. Maka, pemain tidak akan pernah tahu hal apa yang akan ditemui ketika menjelajah sebuah wilayah baru. Terdapat kemungkinan pemain akan menemukan sumber daya yang akan membantu perekonomian atau justru bertemu musuh yang siap menyerang.

Pada Tugas Akhir ini, akan dikembangkan modul Pembangkit Peta Dan Level Dinamis pada *game* Pandawa: the Heroes of Ngastina. Dalam *game* ini, semua aspek pendukung peta tersebut dikemas menjadi satu dalam sebuah level. Tiap level permainan, pemain akan diberikan sebuah peta baru yang berubah-ubah yang dihasilkan secara otomatis oleh Pembangkit Peta dan Level Dinamis. Bagi pemain, hal ini akan membuat *game* semakin menantang dan menambah *experience* pemain. Bagi pengembang,

modul tersebut akan mempermudah pembuatan level dan *world map* secara otomatis.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini, antara lain:

1. Bagaimana membuat wilayah pertarungan yang dinamis dengan menggunakan metode *Cellular Automata*?
2. Bagaimana melakukan pemilihan kombinasi anggota tim musuh dengan mengadaptasi *Coin Changing Problem* dengan menggunakan *Dynamic Programming*?
3. Bagaimana membuat Pembangkit Peta dan Level Dinamis dengan tingkat kesulitan yang bertingkat pada tiap level dengan metode *Elo Rating System*?

1.3 Batasan Permasalahan

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan sebagai berikut :

1. Bentuk peta yang dihasilkan dari Pembangkit Peta dan Level Dinamis tidak mempengaruhi tingkat kesulitan level. Faktor yang mempengaruhi tingkat kesulitan level adalah kombinasi anggota pasukan musuh.
2. Jumlah atribut dari setiap karakter terdapat tiga macam.
3. Jumlah karakter musuh terdiri lima macam.
4. Jumlah level dalam *game* ada sepuluh level.

1.4 Tujuan Tugas Akhir

Tujuan dari pembuatan Tugas Akhir ini adalah membuat modul Pembangkit Peta dan Level Dinamis yang akan diintegrasikan dengan modul lain dalam *game* Pandawa: the Heroes of Ngastina.

1.5 Manfaat Tugas Akhir

Manfaat dari hasil pembuatan Tugas Akhir ini adalah sebagai berikut.

1. Memberikan kemudahan kepada pengembang *game* Pandawa: the Heroes of Ngastina agar dapat membuat level *game* secara otomatis.
2. Menghemat sumber daya dan biaya pengerjaan (*cost development*) karena data-data level tidak perlu disimpan dalam sebuah basis data melainkan dihasilkan secara dinamis.
3. Sebagai modul pendukung dalam *game* Pandawa: the Heroes of Ngastina yang akan diintegrasikan dengan modul lain agar menjadi sebuah *game* yang lengkap.

1.6 Metodologi

Tahapan yang akan dilakukan dalam Tugas Akhir ini di antaranya sebagai berikut.

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dalam pengerjaan Tugas Akhir ini adalah penyusunan proposal Tugas Akhir. Dalam tahap ini, penulis mengajukan gagasan mengenai rancang bangun modul Pembangkit Peta dan Level Dinamis dalam *game* Pandawa: the Heroes of Ngastina. Proposal juga berfungsi sebagai pedoman dalam pengerjaan tugas akhir.

1.6.2 Studi Literatur

Dalam tahap ini dilakukan proses pengumpulan informasi dan literatur yang berhubungan dengan topik Tugas Akhir yang akan dikerjakan. Literatur yang digunakan adalah terkait dengan *Procedural Content Generation (PCG)*, *turn-based strategy game*, *view* isometri, *dynamic level generation*, *Elo Rating System*, *coin-changing problem*, dan lain-lain.

1.6.3 Perancangan

Pada tahap ini dilakukan analisa awal dan pendefinisian kebutuhan sistem *game*. Dari proses tersebut akan dilakukan perumusan rancangan sistem *game*.

1.6.4 Implementasi

Pada tahap ini dilakukan pembuatan *game* sebagai bentuk implementasi dari analisa dan perancangan yang telah dibuat sebelumnya.

1.6.5 Pengujian dan Evaluasi

Tahap ini merupakan tahap pengujian *game* ke beberapa *tester*. Dari umpan balik *tester* akan dipertimbangkan saran-saran yang memungkinkan untuk diterapkan dalam *game*. Dalam tahap ini juga dilakukan pengujian *game* sesuai dengan skenario yang telah disiapkan sebelumnya. Pengujian dan evaluasi dilakukan untuk mencari masalah yang mungkin timbul, mengevaluasi jalannya *game*, dan mengadakan perbaikan jika ada kekurangan.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi *game* yang telah dibuat.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini terdiri dari beberapa bab, yang dijelaskan sebagai berikut.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, permasalahan, batasan masalah, tujuan dan manfaat, metodologi yang digunakan, dan sistematika penyusunan buku Tugas Akhir.

Bab II Tinjauan Pustaka

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan

Bab ini membahas analisa dan perancangan modul *game* yang akan dibangun.

Bab IV Implementasi

Bab ini membahas implementasi dari rancangan sistem yang dilakukan pada tahap perancangan.

Bab V Pengujian dan Evaluasi

Bab ini membahas pengujian dari modul *game* yang dibuat dengan melihat hasil keluaran yang dihasilkan, dan evaluasi untuk mengetahui kemampuan modul.

Bab VI Penutup / Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi kode sumber dan hasil pengujian pada modul ini.

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai dasar teori yang menjadi dasar pembuatan Tugas Akhir ini. Dasar teori yang dibahas di dalam bab ini adalah mengenai flash, Action Script 3.0, view isometri, *Turn-Based Strategy Game, Procedural Content Generation, Dynamic Difficulty Adjustment, ELO Rating System, Cellular Automata, Dynamic Programming, Shockwave Component*, dan *game* serupa yang menggunakan pembangkit level dinamis.

2.1. Flash

Flash adalah salah satu jenis program animasi vektor yang merupakan produk unggulan dari Adobe Systems. *File* yang dihasilkan dari perangkat lunak ini mempunyai ekstensi .swf dan dapat diputar di *browser* yang telah dipasangi Adobe Flash Player. Flash menggunakan bahasa pemrograman bernama Action Script yang muncul pertama kalinya pada Flash 5.

Flash juga dapat digunakan untuk mengembangkan aplikasi-aplikasi web yang kaya dengan pembuatan *script* tingkat lanjut. Di dalam aplikasinya juga tersedia sebuah alat untuk debug *script*. Dengan menggunakan *Code hint* untuk mempermudah dan mempercepat pembuatan dan pengembangan isi Action Script secara otomatis. Untuk memahami keamanan Adobe Flash dapat dilihat dari beberapa sudut pandang, berdasarkan beberapa sumber referensi bahwa tidak ada perbedaan mencolok antara HTML dan JavaScript dimana didalamnya terdapat banyak *tools* yang dapat diambil dari SWF termasuk Action Script. Sehingga kode data dapat terjamin keamanannya. Oleh sebab itu, semua kebutuhan data yang terdapat dalam SWF dapat diambil kembali melalui *server* [15].

2.2. Action Script 3.0

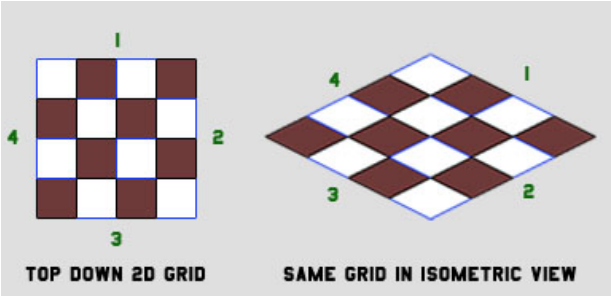
Action Script 3.0 adalah bahasa pemrograman yang dibuat berdasarkan ECMAScript, yang digunakan dalam pengembangan situs web dan perangkat lunak menggunakan *platform* Adobe Flash Player. Action Script juga dipakai pada beberapa aplikasi basis data, seperti Alpha Five. Bahasa ini awalnya dikembangkan oleh Macromedia, tapi kini sudah dimiliki dan dilanjutkan perkembangannya oleh Adobe, yang membeli Macromedia pada tahun 2005.

Action Script terbaru saat ini adalah ActionScript 3.0. Action Script 3.0 adalah bahasa terbaru dari edisi yang sebelumnya dikenal dengan Action Script 2.0. Action Script 3.0 memiliki beberapa kelebihan dibanding pendahulunya, antara lain fitur yang ditawarkan adalah berkas pada Action Script 3.0 dapat dibuat terpisah saat *runtime* [15].

2.3. *View* Isometri

View isometri adalah metode *display* untuk membuat ilusi 3 dimensi pada gambar atau *sprite* 2 dimensi. *View* ini banyak digunakan dalam dunia *game* dan *pixel art*. Dalam *view* isometri, semua objek mempunyai skala yang sama, baik objek jarak jauh maupun jarak dekat. Dalam *view* ini, skala objek akan tetap walaupun kita berpindah tempat dalam *game* tersebut. Sedangkan di dunia nyata, objek akan terlihat semakin kecil ketika kita menjauh dan terlihat semakin besar ketika kita mendekat.

Secara teknis, *view* isometri dapat digambarkan sebagai sebuah kamera yang diletakkan dengan membentuk sudut 45 derajat ke sisi samping kemudian 30 derajat ke sisi bawah. Teknik ini akan membentuk *grid* berbentuk belah ketupat [2]. Beberapa contoh *game* yang menggunakan *view* isometri adalah Age of Empire, Civilization, Final Fantasy Tactics, dan lain-lain.



Gambar 2. 1 Perbandingan *View Top Down* dengan Isometri

Gambar 2. 2 Contoh *Game* dengan *View* Isometri : Age of Empire

2.4. *Turn-Based Strategy Game*

Turn-Based Strategy Game adalah *game* strategi dimana pemain mempunyai giliran untuk melakukan suatu aksi. Pemain dalam *game Turn-Based Strategy* diberikan waktu untuk periode analisis, yang terkadang dibatasi atau tidak dibatasi, untuk memikirkan strategi sebelum melakukan suatu aksi. Pemberian periode waktu ini memungkinkan pemain untuk memberikan

keputusan yang lebih baik pada permainan. Setelah masing-masing pemain mengambil gilirannya, satu putaran permainan akan berakhir kemudian diikuti dengan putaran permainan selanjutnya [16]. Salah satu *game Turn-Based Strategy* yang cukup terkenal adalah *King's Bounty*. *King's Bounty* adalah *game Turn-Based Strategy* yang didesain oleh Jon Van Canegham dari New World Computing pada tahun 1990.

2.5. *Procedural Content Generation* (PCG)

Procedural Content Generation (PCG) adalah teknik pembuatan konten *game* yang terotomatisasi secara algoritmik. Konten yang dimaksud adalah semua aspek dalam *game* yang mempengaruhi *gameplay* kecuali NPC (*Non-Player Character*), misal peta, level, dialog, karakter, *rule-set*, dan senjata. Istilah '*procedural'* mengacu pada prosesnya yang menggunakan fungsi tertentu. Teknik *Procedural Content Generation* biasanya mengacu pada sekumpulan angka yang akan diterjemahkan oleh program untuk membuat konten yang dibutuhkan dalam *game*.

Beberapa keuntungan menggunakan *Procedural Content Generation*, antara lain :

- Dapat mengurangi biaya pembuatan konten *game* karena pembuatan konten *game* secara manual membutuhkan biaya yang besar
- Memungkinkan untuk pengembangan *endless game*
- Dapat mengurangi konsumsi memori karena konten *game* tidak disimpan, melainkan dibangkitkan secara dinamis [3]

2.6. *Dynamic Difficulty Adjustment* (DDA) *in Game*

Dynamic Difficulty Adjustment (DDA), atau disebut juga *Dynamic Game Balancing* (DGB), adalah sebuah teknik untuk mengubah tingkat kesulitan sebuah *game* berdasarkan performa pemain ketika memainkan *game* tersebut. Tujuan dari DDA adalah agar pemain tidak mudah bosan (jika *game* terlalu mudah) atau frustasi (jika *game* terlalu sulit) dan agar dapat memberikan sebuah rangkaian level yang seimbang (*balance*) [5]. Beberapa unsur

dalam *game* yang bisa diubah dengan DDA meliputi : *speed* musuh, *health* musuh, *power* pemain, *power* musuh, dan lain-lain.

Pendekatan yang berbeda ditemukan dalam beberapa literatur tentang DDA. Dalam beberapa kasus, diperlukan pengukuran, baik secara implisit maupun ekspilit, terhadap kesulitan yang dihadapi pemain saat itu. Pengukuran itu dapat dilakukan dengan sebuah fungsi heuristik yang disebut *challenge function*. Fungsi ini memetakan *game state* ke dalam nilai yang menetapkan seberapa mudah atau sulit suatu *game* bagi pemain. Beberapa contoh heuristik yang digunakan, antara lain : jumlah tembakan tepat sasaran, jumlah benda yang berhasil didapatkan, poin nyawa, waktu untuk menyelesaikan *game*, dan lain-lain.

Contoh *game* yang menggunakan DDA adalah *Resident Evil 5* yang dikembangkan pada tahun 2009. *Game* ini mempunyai sebuah sistem yang disebut *Difficulty Scale* yang menilai performa pemain dari skala 1 sampai 10, mengatur perilaku musuh, jenis serangan yang digunakan, dan tingkat kerusakan yang dibuat pemain berdasarkan performa pemain. Tingkat kesulitan yang dipilih mengaitkan pemain pada angka tertentu, sebagai contoh *Normal Difficulty*, pemain yang memulai pada *grade* 4 bisa turun ke *grade* 2 jika memainkan *game* dengan buruk, atau bisa naik ke *grade* 7 jika memainkan *game* dengan baik.

2.7. *ELO Rating System*

ELO Rating System adalah suatu metode untuk menghitung tingkat keterampilan (*skill*) pemain pada permainan yang melibatkan antara dua permain, atau dalam suatu turnamen seperti catur, *American footbal*, *baseball*, dan igo. Sistem ini juga digunakan sebagai peringkat untuk *game multiplayer* pada beberapa jenis *game* komputer. Sistem ini ditemukan oleh Arpad Elo, ahli fisika Amerika Serikat berkebangsaan Hungaria yang sekaligus master catur dan aktif di organisasi Federasi Catur Amerika Serikat (USCF). Pada awalnya, sistem *Elo Rating* dibuat untuk perbaikan pada sistem peringkat olahraga catur dan sudah menjadi standar FIDE (*Fédération Internationale des Échecs*)

yang merupakan Federasi Catur Dunia dalam menghitung *rating* pemain. Tapi sekarang, sistem ini juga digunakan untuk pertandingan olahraga cabang lainnya dan digunakan *Chess Engine*..

Perhitungan *Elo Rating* adalah dengan perhitungan matematika sederhana dengan rumus sebagai berikut :

$$Rn = Ro + K\,(W - We)$$

Rn = *Rating* Baru
Ro = *Rating* Lama
K = Koefisien (Semakin tinggi level pemain, maka nilai koefisien juga semakin tinggi)
W (*Win*) = Nilai (+1 untuk setiap kemenangan, 0 untuk kekalahan)
We (*Win expectation*) = Nilai yang diharapkan berdasarkan *Ro* [13]

2.8. *Cellular Automata*

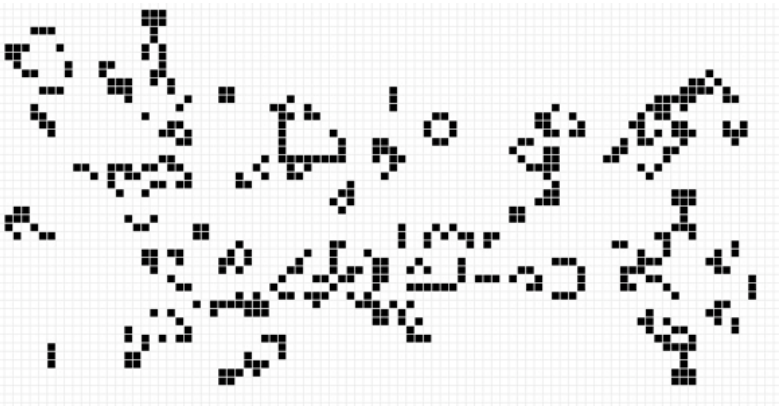
Cellular Automata adalah *grid cell* yang masing-masing mempunyai *state*, seperti *state "on"* dan "*off*" atau "daratan" dan "air". *Cell* tersebut akan mengalami transisi *state* tiap waktu yang akan menghasilkan generasi *cell* baru. Tiap *cell* memiliki aturan (*rule)* untuk menentukan *state* sebagai dasar transisi dari *cell* tersebut dan sel tetangganya. Semua *cell* memiliki *ruleset* yang sama, tapi masing-masing memiliki *state* yang berbeda.

Grid bisa terbentuk dalam jumlah dimensi yang terbatas. Pada tiap *cell*, sekumpulan *cell* yang disebut *neighborhood* (termasuk *cell* itu sendiri) didefinisikan secara relatif pada *cell* tersebut. Sebagai contoh, *neighborhood* dari *cell* dapat didefinisikan sebagai sekumpulan *cell* dengan jarak kurang dari atau sama dengan 2 dari *cell*. Sebuah *initial state* (waktu t = 0) dipilih dengan memberikan *state* pada tiap *cell*. Generasi baru dari *cell* (yang dilanjutkan pada waktu t = 1 dan seterusnya) dibuat berdasarkan aturan yang telah ditetapkan yang menentukan state

baru pada tiap *cell* yang meliputi state dari *cell* tersebut dan *state cell* tetangga.

Contoh penerapan *Cellular Automata* adalah Conway's Game of Life atau disebut juga Game of Life. Game of Life adalah sebuah *cellular automata* yang ditemukan oleh matematikawan Inggris, John Horton Conway, pada tahun 1970. Tiap *cell* dalam Game of Life mempunyai *state* hidup atau mati. Tiap *cell* berinteraksi dengan delapan tetangganya, yaitu *cell* yang berbatasan langsung secara horisontal, vertikal, atau diagonal. Aturan transisi dalam Game of Life adalah :

a. Setiap *cell* hidup dengan kurang dari dua tetangga hidup maka cell tersebut akan mati, seolah-olah disebabkan oleh kekurangan penduduk.
b. Setiap *cell* hidup dengan dua atau tiga tetangga hidup maka *cell* tersebut akan hidup ke generasi berikutnya.
c. Setiap sel hidup dengan lebih dari tiga tetangga hidup mati, seolah-olah karena hidup dengan berdesak-desakan.
d. Setiap sel mati dengan tepat tiga tetangga hidup menjadi sel hidup, seolah-olah telah terjadi reproduksi [14].



Gambar 2. 3 Game of Life, contoh *Cellular Automata* sederhana

2.9. *Dynamic Programming*

Dynamic Programming adalah teknik perancangan algoritma yang dikembangkan untuk menyelesaikan permasalahan yang sangat kompleks dengan memecah permasalahan tersebut menjadi beberapa sub-permasalahan. Permasalahan tersebut dipecah menjadi sekumpulan langkah (*step*) dan tahapan (*stage*). sedemikian sehingga solusi dari permasalahan tersebut dapat dipandang dari serangkaian keputusan yang saling berkaitan. *Dynamic Programming* biasanya digunakan dalam masalah optimasi Langkah-langkah penyelesaian permasalahan dengan metode ini adalah :

1. Karakteristikkan struktur solusi optimal
2. Definisikan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Konstruksi solusi optimal [8].

2.10. *Coin-Changing Problem*

Coin-Changing Problem adalah persoalan untuk menemukan cara untuk menukar sejumlah uang tertentu, *n,* dengan sekumpulan pecahan mata uang tertentu, ***d1***….***dm***. Contoh persoalan *Coin-Changing Problem* adalah : Di sebuah pusat perbelanjaan, seorang kasir perlu memberikan kembalian kepada pembeli setelah terjadi transaksi pembelian. Uang kembalian tersebut terdiri dari beberapa satuan pecahan mata uang tertentu $ = (c1, c2, ...cm) dimana 1 = c1, c2 < ... < cm. Sebagai contoh, terdapat empat jenis pecahan mata uang yaitu $ = (1, 5, 10, 25). Dikarenakan jumlah koin yang diperlukan sangat terbatas, maka kasir tersebut harus memberikan pecahan kembalian koin tersebut dengan jumlah seminimal mungkin [6].

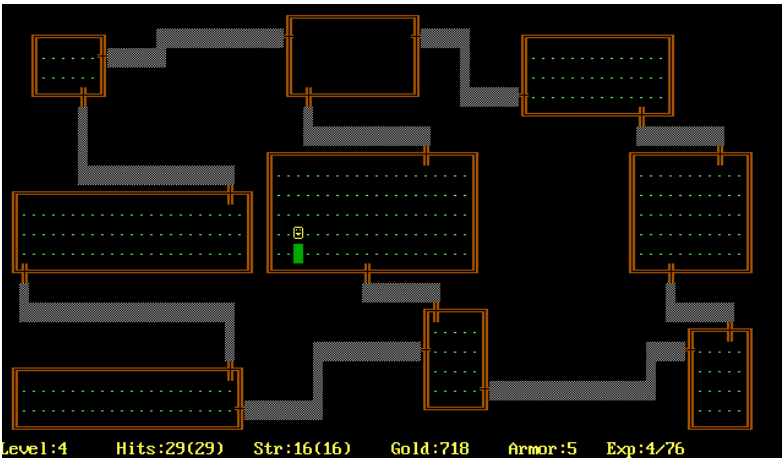
2.11. *Game* Serupa yang Menggunakan Pembangkit Level Dinamis

Game serupa yang menggunakan pembangkit level dinamis adalah Rogue. *Game* ini adalah *single player game* yang merupakan pendahulu dari semua *game roguelike. Game* ini juga

merupakan contoh paling awal dari *game* yang menggunakan *Procedural Content Generation* yang dikembangkan oleh Michael Toy dan Glenn Wichman pada tahun 1980.

Rogue adalah *game* dengan tipe *dungeon crawling* dimana pemain harus mengelilingi labirin *dungeon*, melawan monster, dan mengumpulkan harta karun. *Game* ini juga dilengkapi fitur *permadeath* (*permanent death*), yaitu sekali pemain mati, maka pemain harus mengulang *game* dari awal lagi. Dengan menggunakan PCG, pemain bisa menggunakan *skill* yang ia dapatkan dari *game* tanpa mengalami repetisi dari level *game* yang sama yang dimainkan berulang [17].

.



Gambar 2.4 *game* Rogue asli yang menggunakan PCG

BAB III
ANALISIS DAN PERANCANGAN

Bab ini membahas tahap analisis permasalahan dan perancangan dari sistem yang akan dibangun dalam Tugas Akhir ini. Analisis permasalahan membahas permasalahan yang diangkat dalam pengerjaan Tugas Akhir. Perancangan sistem membahas tentang rancangan sistem berdasarkan hasil analisis.

3.1. Analisis

Tahap analisis dibagi menjadi beberapa bagian antara lain analisis permasalahan, deskripsi umum modul, perancangan sistem.

3.1.1. Analisis Permasalahan

Permasalahan utama yang diangkat dalam pembuatan Tugas Akhir ini adalah bagaimana membuat peta dan level permainan pada *game* Pandawa:the Heroes of Ngastina secara dinamis dengan dengan tingkat kesulitan yang menyesuaikan kemampuan pemain. Bagi pemain, hal ini akan membuat *game* semakin menantang dan menambah *experience* pemain. Bagi pengembang, Pembangkit Peta dan Level Dinamis ini akan mempermudah pembuatan level dan *world map* secara otomatis.

3.1.2. Deskripsi Umum Modul

Game Pandawa: the Heroes of Ngastina adalah *game* dengan genre *Turn-Based Strategy* yang ditampilkan dalam *view* isometri. *Game* ini mengangkat salah satu cerita budaya Indonesia yaitu Pandawa Lima sebagai lakon utama dari permainan. Pemain berperan sebagai komandan pasukan Pandawa yang memberikan komando pada pasukan untuk melakukan aksi tertentu melalui perintah suara dengan teknologi *voice recognition*.

Secara garis besar, terdapat tiga hal inti pada tiap level *game* ini, yaitu peta, musuh, dan aturan. Atribut peta meliputi bentuk peta permainan dan nama lokasi. Atribut musuh meliputi

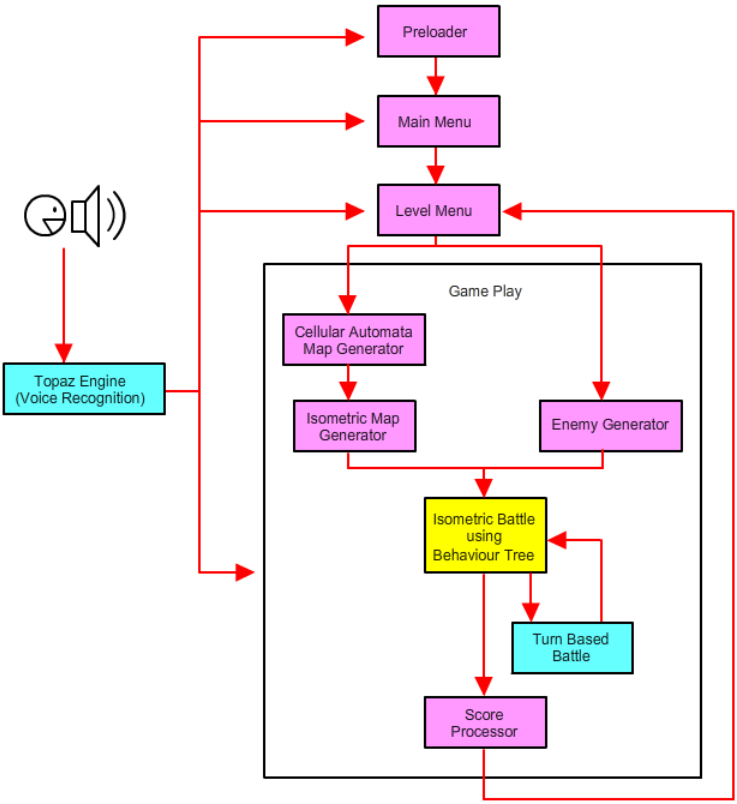
jumlah anggota pasukan musuh, kombinasi anggota pasukan musuh, kekuatan masing-masing anggota, status masing-masing anggota, dan posisi musuh. Aturan meliputi kondisi menang dan kalah pada sebuah level.

Game Pandawa: the Heroes of Ngastina dibangun dengan bahasa pemrograman Action Script 3.0 dengan menggunakan IDE Adobe Flash CC. *Game* ini terdiri dari tiga buah modul utama yang akan mendukung *gameplay*, antara lain : Modul Voice Recognition untuk memberikan perintah suara kepada pasukan Pandawa, Modul Pembangkit Peta dan Level Dinamis untuk membuat level dan *world map* secara otomatis, dan Modul Kecerdasan Buatan Perilaku Karakter untuk mengatur perilaku pemain dan lawan. Dalam Tugas Akhir ini, penulis akan mengembangkan Modul Pembangkit Peta dan Level Dinamis yang akan diintegrasikan dengan modul lainnya dalam *game* Pandawa: the Heroes of Ngastina.

3.1.3. Diagram Alur *Game* Pandawa:the Heroes of Ngastina

Diagram alur *game* Pandawa: the Heroes of Ngastina dapat dilihat pada gambar 3.1. *Game* ini terdiri dari tiga buah modul utama, yaitu Modul Voice Recognition, Modul Pembangkit Peta dan Level Dinamis, dan Modul Kecerdasan Buatan Perilaku Karakter. Tugas Akhir ini akan fokus kepada pengembangan modul Pembangkit Peta dan Level Dinamis yang meliputi Submodul Cellular Automata Map Generator, Submodul Isometric Map Generator, Submodul Enemy Generator, dan Submodul Score Processor.

Ketika *game* telah menampilkan Preloader untuk memuat *file game* dan menampilkan Main Menu, maka *game* akan menampilkan Level Menu yang terdiri dari sepuluh peta level permainan. Dari Level Menu, pemain akan diarahkan ke *gameplay* yang di dalamnya terdapat Modul Pembangkit Peta dan Level Dinamis.



Gambar 3. 1 Diagram Alur *Game* Pandawa:the Heroes of Ngastina

Alur kerja Modul Pembangkit Peta dan Level Dinamis ini adalah sebagai berikut.

1. Ketika *gameplay* dibuka, secara otomatis Submodul Cellular Automata Map Generator akan membuat peta Cellular Automata.
2. Setelah peta Cellular Automata dibuat, Submodul Isometric Map Generator dan Submodul Enemy Generator secara

bersamaan akan membentuk komponen penyusun *stage* permainan secara dinamis. Submodul Isometric Map Generator akan membentuk peta isometri. Submodul Enemy Generator akan membentuk kombinasi musuh.

3. Setelah pemain menyelesaikan suatu level *game*, maka kemampuan pemain akan dinilai dengan beberapa kriteria pada Submodul Score Processor. Submodul ini akan memberikan keluaran berupa *rating* pemain yang akan menjadi nilai untuk membangkitkan kombinasi musuh pada level selanjutnya.
4. Hasil *rating* pemain tersebut kemudian akan dikirimkan kembali ke Level Menu, kemudian dari trigger *start game* dan masukan dari Level Menu tersebut akan terbentuk peta dan level selanjutnya.

3.2. Perancangan

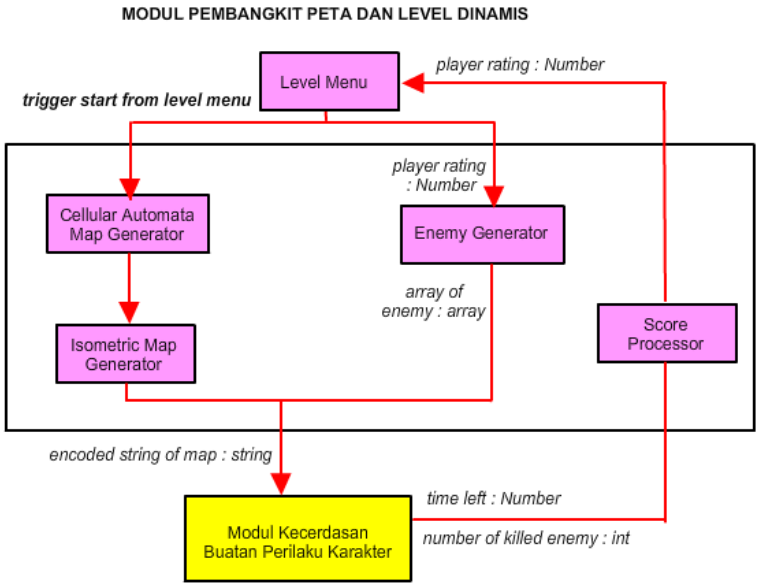
Penjelasan tahap perancangan dalam sistem ini dibagi menjadi beberapa bagian yaitu perancangan sistem modul, perancangan submodul, dan perancangan antarmuka.

3.2.1. Perancangan Sistem Modul

Modul Pembangkit Peta dan Level Dinamis dibangun dengan mengimplementasikan teknik *Procedural Content Generation* (PCG) dalam pembuatan level dan *world map*. Level dan *world map* akan dibuat secara algoritmik sehingga pengembang tidak perlu membuatnya secara manual. Pembangkit Peta dan Level Dinamis akan membangkitkan sebuah level dan *world map* baru yang berubah-ubah tiap kali *game* dimainkan. Level *game* dibangkitkan sesuai dengan kemampuan pemain yang dievaluasi ketika pemain memainkan suatu level. Hal ini tentu akan membuat pemain tidak mudah bosan serta dapat menambah *experience* pemain dalam memainkan *game* ini.

Diagram Modul Pembangkit Peta dan Level Dinamis yang ditunjukkan pada Gambar 3.2 menggambarkan hubungan antara empat buah submodul dalam modul Pembangkit Peta dan Level

Dinamis. Empat buah submodul tersebut terdiri dari Submodul Cellular Automata Map Generator, Submodul Isometric Map Generator, Submodul Score Processor, dan Submodul Enemy Generator.



Gambar 3. 2 Diagram Modul Pembangkit Peta dan Level Dinamis

Di awal permainan, Modul Pembangkit Peta dan Level Dinamis menerima masukan *rating* pemain dan *trigger Start Game* dari pemain. Modul ini memberikan keluaran berupa *string* peta, dan *array* kombinasi anggota tim musuh yang menentukan tingkat kesulitan suatu level. Sedangkan di akhir permainan, modul ini menerima masukan berupa sisa waktu permainan dan jumlah musuh yang terbunuh. Masukan tersebut diolah menjadi keluaran berupa *rating* pemain. Keluaran tersebut kemudian dikirimkan kembali ke Level Menu untuk membangkitkan level selanjutnya.

3.2.1.1. Perancangan Submodul Cellular Automata Map Generator

Submodul Cellular Automata Map Generator adalah submodul yang menangani proses pembentukan *string* pembentuk peta isometri dalam *game*. Pembentukan peta dalam submodul ini menggunakan metode *Cellular Automata*. Metode ini dimulai dengan pembuatan *tile* tunggal yang akan membentuk *grid* peta. Tiap *tile* menyimpan informasi tentang objek apa yang akan dipasang pada *tile* tersebut yang direpresentasikan dengan *id* jenis *tile*. Terdapat enam jenis objek dalam *game* ini, antara lain :

Tabel 3. 1 Tabel Daftar Jenis Objek *Tile*

ID *Tile*	Jenis Objek	Gambar *Tile*
1	*Tile* Rumput	
2	*Tile* Hutan	
3	*Tile* Batu	
4	*Tile* Pohon	
5	*Tile* Pemain	
6	*Tile* Musuh	

Peta permainan dibuat dengan ukuran 30 x 20, *tile* tunggal dipasang pada *grid* sebanyak 30 *tile* mendatar dan 20 *tile* menurun. Jenis *tile* yang dipasang ditentukan oleh persentase luas area rumput pada peta, yaitu 70% atau 0,7. Jika hasil pengacakan nilai pada suatu *tile* kurang dari nilai persentase, maka *tile* yang akan dipasang adalah *tile* rumput. Jika pengacakan nilai pada suatu *tile* lebih dari nilai persentase, maka *tile* yang akan dipasang adalah *tile* hutan. Pada tahap ini, *tile-tile* pada peta masih dipasang secara tak beraturan dan belum bisa dijadikan sebagai area pertarungan.

Langkah selanjutnya yang akan dilakukan adalah proses transisi *state* dari semua *tile* yang telah dipasang pada *grid* peta. *State* dari *tile* tetangga direpresentasikan dengan *neighborhood value* yang dihitung dari jumlah *tile* tertentu (misal, *tile* rumput) di sekitar *tile* yang ditentukan. *Central base grid* diinisialisasi oleh *tile* dengan tipe tertentu. Kemudian, sebagian *tile* dalam *grid* tersebut akan diubah menjadi *tile* dengan tipe lain sesuai dengan jumlah persentase jenis *tile* yang akan diletakkan dalam peta dan akan dibentuk sesuai dengan aturan yang telah ditentukan.

Proses transisi *Cellular Automata* pada peta ini dilakukan dengan langkah-langkah sebagai berikut :

Tabel 3. 2 Tahapan Pembentukan Peta *Cellular Automata* pada *Game* Pandawa: the Heroes of Ngastina

Tahap	Proses
Inisialisasi	Inisialisasi basis peta dengan memasang *tile* sebanyak 30 x 20. Jenis *tile* yang dipasang pada tahap ini hanya *tile* rumput dan *tile* hutan
Iterasi 1 – 5	Proses pembentukan area rumput dan hutan agar bisa menjadi area pertarungan (dengan aturan pembentukan peta nomor 1 dan 2)
Iterasi 6	Proses penempatan halangan dalam permainan, berupa pohon dan batu (dengan aturan pembentukan peta nomor 3)
Iterasi 7	Proses penempatan pemain dan musuh dalam peta (dengan aturan pembentukan peta nomor 4)

Pada tiap proses di atas akan dilakukan transisi *state* pada tiap *tile* sesuai dengan *rule*. *Rule* pembentuk peta yang didefinisikan dalam *game* Pandawa:the Heroes of Ngastina adalah sebagai berikut.

Tabel 3. 3 *Rule* Pembentukan Peta Isometri pada *Game* Pandawa:the Heroes of Ngastina

No.	Keadaan awal	Aturan
1.	Inisialisasi peta	Pada tahap inisialisasi, peta hanya terdiri dari *tile* rumput dan hutan sebagai basis awal peta. Pada tahap iterasi 1 – 5, akan dilakukan pembentukan ulang area basis peta tersebut. Jika *tile* rumput dikelilingi oleh minimal empat *tile* rumput di sekitarnya, maka *tile* tersebut akan tetap menjadi *tile* rumput. Sebaliknya, jika jumlah *tile* rumput di sekitarnya kurang dari empat, maka *tile* tersebut akan diubah menjadi *tile* hutan.
2.	Inisialisasi peta	Pada tahap iterasi 1 – 5, jika *tile* hutan dikelilingi oleh minimal lima *tile* rumput, maka *tile* tersebut akan diubah menjadi *tile* rumput. Jika jumlah *tile* rumput di sekitarnya kurang dari lima, maka *tile* hutan tersebut tidak berubah.
3.	Basis peta telah selesai dibuat	Jika suatu *tile* adalah *tile* rumput, maka terdapat dua kemungkinan *tile* tersebut akan menjadi *tile* rumput atau *tile* pohon.
4.	Basis peta telah selesai dibuat dan halangan pada peta telah ditempatkan	*Tile* pemain dan *tile* musuh ditempatkan berjajar dengan jarak lima kotak dari tepi *stage* permainan. *Tile-tile* pemain berada di sebelah kiri, sedangkan *tile-tile* musuh berada di sebelah kanan.

No.	Keadaan awal	Aturan
		Jika *tile* yang akan diubah adalah sebuah *tile* rumput, maka *tile* tersebut bisa diubah menjadi *tile* pemain atau *tile* musuh. *Tile* batu dan pohon tidak bisa diubah menjadi *tile* pemain atau musuh.

Ketika peta *Cellular Automata* telah selesai dibuat, maka akan dibentuk sebuah *string* pembentuk peta tersebut. *String* tersebut menampung data properti peta dan data semua objek yang dipasang pada tiap *tile* peta. Kedua data tersebut dipisahkan dengan karakter pemisah atau *splitter* ">".

Data properti peta terdiri dari ukuran *tile*, panjang peta, dan lebar peta dengan karakter pemisah "#". Sedangkan data objek pada peta terdiri dari karakter angka yang merepresentasikan id objek pada semua *tile* dengan karakter pemisah (*splitter*) berupa karakter kosong. Tampilan format *string* pembentuk peta tersebut adalah sebagai berikut :

```
[ukuran_tile]#[panjang_peta]#[lebar_peta]>
[id_objek_baris_1_kolom_1][id_objek_baris_1_kolom_
2]......[id_objek_baris_n_kolom_n]
```

Contoh *string* pembentuk peta :

```
70#30#20>121132....3
```

Panjang *string* pembentuk peta yang dihasilkan dari peta *Cellular Automata* ini berukuran 609 karakter. *String* tersebut terdiri dari 9 karakter data properti peta dengan karakter pemisah atau *splitter* dan 600 karakter angka yang mewakili tiap objek pada peta berukuran 30 x 20. Keluaran tersebut akan menjadi masukan untuk Submodul Isometric Map Generator dan Modul Kecerdasan Buatan Perilaku Karakter untuk menghasilkan peta level selanjutnya.

3.2.1.2. Perancangan Submodul Isometric Map Generator

Submodul Isometric Map Generator adalah submodul yang menangani pembentukan peta isometri dalam *game* Pandawa: the Heroes of Ngastina. Submodul ini membutuhkan masukan berupa *string* pembentuk peta (*map*) dari Submodul Cellular Automata Map Generator. *String* tersebut kemudian dikonversi menjadi *array* 2 dimensi sesuai dengan karakter pemisah atau *splitter* yang telah ditentukan.

Beberapa jenis karakter pemisah dalam *string* tersebut adalah :

- > : karakter pemisah antara data properti peta dengan data semua *tile* peta
- # : karakter pemisah antara tiap data properti peta yang terdiri dari ukuran *tile*, panjang peta, dan lebar peta
- **[]** : karakter kosong sebagai pemisah data tiap *tile* pada peta
- Contoh *string* pembentuk peta dari Submodul Cellular Automata Map Generator adalah sebagai berikut :

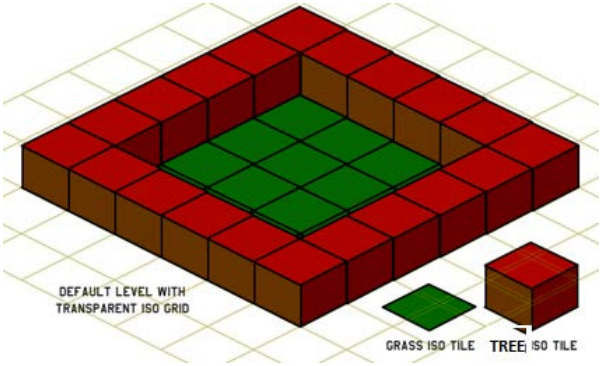
```
60#30#20>121132....3
```

Array tersebut memiliki dimensi *m x n* sesuai dengan jumlah baris dan kolom pada peta. Di bawah ini adalah contoh *array* 2 dimensi berukuran 6 x 6 untuk peta isometri.

```
[[3,3,3,3,3,3],
 [3,2,2,2,2,3],
 [3,2,2,2,2,3],
 [3,2,2,2,2,3],
 [3,2,2,2,2,3],
 [3,3,3,3,3,3]]
```

Dari *array* 2 dimensi di atas, maka enam buah *tile* tunggal akan dipasang dengan posisi enam mendatar dan enam menurun membentuk sebuah peta seluas (6 x ukuran *tile*) x (6 x ukuran *tile*). Satuan ukuran *tile* pada Submodul Isometric Map Generator adalah

pixel. Jika ukuran tiap *tile* tunggal adalah 70 *pixel*, maka luas peta isometri 6 x 6 tersebut adalah 176.400 *pixel*. Ilustrasi peta dalam bentuk isometri dari masukan di atas adalah sebagai berikut.



Gambar 3. 3 Penggambaran Peta Isometri Berukuran 6 x 6

3.2.1.3. Perancangan Submodul Score Processor

Submodul Score Processor adalah submodul yang menangani proses evaluasi atau penilaian kemampuan pemain dalam memainkan suatu level. Kemampuan pemain dievaluasi dengan dua jenis kriteria yang menjadi masukan untuk submodul ini, antara lain sisa waktu dalam menyelesaikan level (*time left* : *integer*) dan jumlah musuh yang terbunuh (*number of killed enemy* : *integer*). Perhitungan skor menggunakan rumus sebagai berikut.

$$score = level\ score + (C_1 \times time\ left + C_2 \times number\ of\ killed\ enemy)$$

score = total skor pemain
level score = skor pada suatu level (level x 150)
time left = sisa waktu permainan
number of killed enemy = jumlah musuh yang terbunuh
C1 = konstanta *time left* (dengan nilai 0,35)
C2 = konstanta *number of killed enemy* (dengan nilai 0,65)

Setelah skor didapatkan, maka akan dilakukan penghitungan *rating* pemain dengan menggunakan *Elo Rating System. Rating* pemain berfungsi sebagai nilai modal yang menjadi masukan untuk Submodul Enemy Generator. Nilai modal digunakan untuk membentuk kombinasi pasukan musuh yang sesuai dengan kemampuan pemain pada level selanjutnya. Rumus untuk menghitung *rating* pemain adalah sebagai berikut.

$$Rn = Ro + K\,(W - We)$$

Rn = *Rating* Baru Pemain
Ro = *Rating* Lama Pemain (nilai *default* 160)
K = Koefisien (nilai koefisien 100)
W (*Win*) = Nilai (+1 untuk setiap kemenangan, 0 untuk kekalahan)
We (*Win expectation*) = Nilai yang diharapkan berdasarkan *Ro*. Nilai ini didapat dari skor yang merupakan masukan dari Submodul Score Processor.

Nilai *We* didapatkan dari standar FIDE *Rating* yang bisa dilihat pada Gambar 3.4.

D	P_D		D	P_D		D	P_D		D	P_D	
Rtg Dif	H	L	Rtg Dif	H	L	Rtg Dif	H	L	Rtg Dif	H	L
0–3	.50	.50	92–98	.63	.37	198–206	.76	.24	345–357	.89	.11
4–10	.51	.49	99–106	.64	.36	207–215	.77	.23	358–374	.90	.10
11–17	.52	.48	107–113	.65	.35	216–225	.78	.22	375–391	.91	.09
18–25	.53	.47	114–121	.66	.34	226–235	.79	.21	392–411	.92	.08
26–32	.54	.46	122–129	.67	.33	236–245	.80	.20	412–432	.93	.07
33–39	.55	.45	130–137	.68	.32	246–256	.81	.19	433–456	.94	.06
40–46	.56	.44	138–145	.69	.31	257–267	.82	.18	457–484	.95	.05
47–53	.57	.43	146–153	.70	.30	268–278	.83	.17	485–517	.96	.04
54–61	.58	.42	154–162	.71	.29	279–290	.84	.16	518–559	.97	.03
62–68	.59	.41	163–170	.72	.28	291–302	.85	.15	560–619	.98	.02
69–76	.60	.40	171–179	.73	.27	303–315	.86	.14	620–735	.99	.01
77–83	.61	.39	180–188	.74	.26	316–328	.87	.13	over735	1.0	.00
84–91	.62	.38	189–197	.75	.25	329–344	.88	.12			

Gambar 3. 4 Nilai *We* Berdasarkan FIDE *Rating*

3.2.1.4. Perancangan Submodul Enemy Generator

Submodul Enemy Generator adalah submodul yang menangani proses pembentukan kombinasi anggota tim musuh pada suatu level berdasarkan nilai modal dari Submodul Score Processor. Nilai modal merupakan nilai batas maksimum untuk memilih kombinasi anggota pasukan musuh pada tiap level. Contoh nilai modal pada tiap level ditunjukkan pada tabel di bawah ini :

Tabel 3. 4 Tabel Contoh Nilai Modal Tiap Level

Level	Nilai Modal
1	160
2	244
3	343
4	443
5	543
6	643
7	743
8	843
9	943
10	1043

Kombinasi tim musuh yang dihasilkan akan menyesuaikan kemampuan pemain berdasarkan nilai modal pada tiap level. Terdapat lima jenis musuh pada *game* ini dengan kemampuan yang berbeda-beda. Tabel 3.5 menunjukkan profil dari tiap musuh dengan detail status kekuatannya masing-masing yang terdiri dari nilai *health, power,* dan *speed.* Ketiga nilai tersebut akan dijumlahkan sehingga menghasilkan nilai *enemy grade* yang menjadi nilai total kekuatan tiap musuh.

Tabel 3. 5 Tabel Status Kemampuan Musuh

Nama Musuh	Tampilan Karakter	Status
Sengkuni (*id enemy* = 1)		*Health* : 30 *Power* : 10 *Speed* : 40 *Enemy Grade :* 80
Dursasana (*id enemy* = 2)		*Health* : 50 *Power* : 50 *Speed : 40* *Enemy Grade :* 140
Karna (*id enemy* = 3)		*Health* : 80 *Power* : 120 *Speed :* 80 *Enemy Grade :* 200

Nama Musuh	Tampilan Karakter	Status
Duryudana (*id enemy* = 4)		*Health* : 80 *Power* : 110 *Speed* : 60 *Enemy Grade :* 250
Bogadenta (*id enemy* = 5)		*Health* : 100 *Power* : 250 *Speed : 50* *Enemy Grade :* 300

Pada level pertama, nilai modal diinisialisasi sebesar 160 dan kombinasi musuh yang dihasilkan adalah dua orang musuh dengan jenis Sengkuni. Pada level selanjutnya, kombinasi musuh yang akan dihadapi antara satu pemain dengan pemain lainnya bisa berbeda. Pemilihan anggota tim musuh mengadaptasi *Coin-Changing Problem* dengan penyelesaian menggunakan *Dynamic Programming*. *Enemy grade* dari kelima musuh diasumsikan sebagai koin dengan nilai satuan antara lain 80, 140, 200, 250, dan 300. Nilai modal diasumsikan sebagai uang yang akan ditukarkan dengan beberapa pemain, yang jika nilai satuannya dijumlahkan maka hasilnya kurang dari atau sama dengan nilai modal tersebut.

Di bawah ini merupakan *pseudocode Coin-Changing Problem* dengan algoritma *Dynamic Programming*.

Masukan : d (*array enemy grade*), k (jumlah musuh), n (nilai modal yang akan ditukar)

Keluaran : C (jumlah musuh) dan S (indeks musuh dalam *array enemy grade*)

```
CHANGE(d, k, n)
1  C[0] ← 0
2  for p ← 1 to n
3          min ← ∞
4          for i ← 1 to k
5              if d[i] ≤ p then
6                  if 1 + C[p − d[i]] < min then
7                      min ← 1 + C[p − d[i]]
8                      coin ← i
9          C[p] ← min
10         S[p] ← coin
11 return C and S
```

Gambar 3. 5 Algoritma *Dynamic Programming*

Keluaran dari submodul ini adalah *array* kombinasi id tim musuh yang akan menjadi masukan untuk modul lain, yaitu Modul Kecerdasan Buatan Perilaku Karakter. Jumlah musuh yang muncul pada tiap level dibatasi hanya lima orang. Satu jenis musuh bisa muncul lebih dari satu pada suatu level.

3.2.2. Perancangan Kelas

Pada subbab ini akan dijelaskan mengenai rancangan kelas yang akan digunakan dalam program *game* Pandawa:the Heroes of Ngastina sesuai dengan tiap submodul.

3.2.2.1. Perancangan Kelas pada Submodul Cellular Automata Map Generator

Pada submodul ini diperlukan dua buah kelas, yaitu kelas TileCA, dan kelas CellularAutomata. Rancangan masing-masing kelas tersebut adalah sebagai berikut.

a. Kelas TileCA

Kelas ini merupakan representasi dari *tile* tunggal yang membentuk peta *cellular automata* pada Submodul Cellular Automata Map Generator. Kelas ini mempunyai beberapa atribut sebagai berikut.

Tabel 3. 6 Rancangan Kelas TileCA

Nama Atribut	**Keterangan**
IdTile	id *tile* sesuai dengan objek yang dipasang pada *tile* tersebut
TileWidth	Ukuran *tile*

b. Kelas CellularAutomata

Kelas ini adalah kelas yang digunakan untuk memasang semua *tile* tunggal ke dalam peta dan melakukan proses iterasi *cellular automata* sesuai dengan aturan peta *game* yang telah ditentukan. Kelas ini mempunyai beberapa atribut sebagai berikut.

Tabel 3. 7 Rancangan Kelas CellularAutomata

Nama Atribut	**Keterangan**
LandRatio	Persentase perbandingan antara *tile* hutan dan *tile* rumput pada peta sebagai basis *tile* pembentuk peta
ObstacleCoverage	Persentase banyaknya halangan pada peta *game*
NumberOfEnemies	Jumlah musuh pada peta suatu level
StringOfArray2DMap	Hasil konversi array 2 dimensi peta ke dalam bentuk *string*

3.2.2.2. Perancangan Kelas pada Submodul Isometric Map Generator

Pada submodul ini diperlukan dua buah kelas, yaitu kelas Tile dan TileManager. Rancangan masing-masing kelas tersebut adalah sebagai berikut.

a. Kelas Tile

Kelas ini merupakan representasi dari *tile* tunggal yang membentuk peta isometri pada Submodul Isometric Map Generator. Kelas ini mempunyai beberapa atribut sebagai berikut.

Tabel 3. 8 Rancangan Kelas Tile

Nama Atribut	**Keterangan**
IdColour	id warna *tile* sesuai dengan objek yang dipasang pada *tile* tersebut
TileWidth	Ukuran *tile.* Dari ukuran *tile* ini akan dibentuk *tile* isometri dengan lebar = tileWidth, dan tinggi = 0,5 x tileWidth
IdRow	Nomor baris *tile* pada peta
IdColumn	Nomor kolom *tile* pada peta

b. Kelas TileManager

Kelas TileManager adalah kelas yang berfungsi untuk menyusun semua *tile* tunggal menjadi peta isometri. Di dalam kelas ini juga dilakukan pengaturan posisi dan warna tiap *tile*. Kelas TileManager memiliki beberapa atribut sebagai berikut.

Tabel 3. 9 Rancangan Kelas TileManager

Nama Atribut	**Keterangan**
ArrayHeight	Dimensi panjang peta isometri
ArrayWidth	Dimensi lebar peta isometri
TileWidth	Ukuran *tile* yang akan dipasang pada peta isometri
InitXStart	Posisi *x* koordinat awal peta
InitYStart	Posisi *y* koordinat awal peta

3.2.2.3. Perancangan Kelas pada Submodul ScoreProcessor

Pada submodul ini diperlukan dua buah kelas, yaitu kelas ScoreProcessor dan kelas EloRating. Rancangan masing-masing kelas tersebut adalah sebagai berikut.

a. Kelas ScoreProcessor

Kelas ScoreProcessor adalah kelas yang berfungsi untuk mengkonversi dua parameter yaitu sisa waktu suatu level dan jumlah musuh yang terbunuh menjadi skor. Kelas ScoreProcessor mempunyai atribut sebagai berikut.

Tabel 3. 10 Rancangan Kelas ScoreProcessor

Nama Atribut	**Keterangan**
numberOfKilledEnemy	Jumlah musuh yang terbunuh pada suatu level
Time Left	Sisa waktu permainan pada suatu level
Constanta1	Nilai konstanta *time left*
Constanta2	Nilai konstanta *numberOfKilledEnemy*

b. Kelas EloRating

Kelas ScoreProcessor adalah kelas yang berfungsi untuk mengolah skor pemain menjadi nilai modal yang akan digunakan untuk membangkitkan kombinasi musuh pada level selanjutnya sesuai kemampuan pemain. Kelas EloRating mempunyai atribut sebagai berikut.

Tabel 3. 11 Rancangan Kelas EloRating

Nama Atribut	**Keterangan**
OldRating	*Rating* lama pemain
NewRating	*Rating* baru pemain yang akan menjadi nilai modal untuk level selanjutnya

3.2.2.4. Perancangan Kelas pada Submodul Enemy Generator

Pada submodul ini diperlukan dua buah kelas, yaitu kelas Enemy dan kelas CoinChangingProblem. Rancangan masing-masing kelas tersebut adalah sebagai berikut.

a. Kelas Enemy

Kelas Enemy merupakan representasi musuh dalam *game* yang akan dihadapi pemain. Kelas ini mempunyai atribut sebagai berikut.

Tabel 3. 12 Rancangan Kelas Enemy

Nama Atribut	**Keterangan**
IdEnemy	Id jenis musuh
EnemyGrade	Nilai bobot kekuatan musuh

b. Kelas CoinChangingProblem

Kelas Enemy merupakan representasi musuh dalam *game* yang akan dihadapi pemain. Kelas ini mempunyai atribut sebagai berikut.

Tabel 3. 13 Rancangan Kelas CoinChangingProblem

Nama Atribut	**Keterangan**
ArrayOfDenomination	*Array* yang menampung kekuatan semua musuh dalam *game* sebagai satuan untuk menukar nilai modal
PlayerRating	Nilai modal pemain
ArrayOfEnemyCombination	*Array* kombinasi musuh sesuai dengan nilai modal pemain

3.2.3. Perancangan Antarmuka

Pada subbab ini akan dijelaskan mengenai rancangan antarmuka *game* Pandawa:the Heroes of Ngastina. Pada subbab ini akan dijelaskan perancangan antarmuka *main menu*, antarmuka *level menu*, dan antarmuka profil musuh.

3.2.3.1. Perancangan Antarmuka Main Menu

Antarmuka *main menu game* Pandawa:the Heroes of Ngastina merupakan tampilan menu utama *game* Pandawa:the Heroes of Ngastina. *Main Menu* akan menampilkan judul game dan menu awal permainan, yaitu menu "Mainkan" untuk menuju ke *gameplay*.

3.2.3.2. Perancangan Antarmuka Tampilan Level Menu

Antarmuka level menu merupakan tampilan keseluruhan peta permainan *game* Pandawa:the Heroes of Ngastina. Terdapat sepuluh area sekaligus level yang akan dimainkan pemain. Pada tampilan level menu, semua area akan terkunci kecuali area pada level satu. Pada level menu terdapat menu "Kembali" untuk kembali ke menu utama dan "Musuh" untuk melihat profil musuh.

3.2.3.3. Antarmuka Tampilan Profil Musuh

Antarmuka profil musuh menunjukkan daftar musuh dalam *game*. Tampilan antarmuka ini akan dibagi menjadi dua *tab* kiri dan kanan. *Tab* kiri menunjukkan daftar musuh, yaitu Sengkuni, Dursasana, Karna, Duryudana, dan Bogadenta. *Tab* kanan menunjukkan profil masing-masing musuh yang meliputi *health, power, speed,* dan *enemy grade.*

BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Bahasa pemrograman yang digunakan adalah bahasa pemrograman Action Script 3.0.

4.1. Lingkungan Implementasi

Dalam merancang perangkat lunak ini digunakan beberapa perangkat pendukung sebagai berikut.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah komputer. Spesifikasi dari perangkat tersebut adalah laptop Asus Ultrabook S400, Intel Core i5, 8 GHz dan 4 GB *Memory*.

4.1.2 Lingkungan Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut.

Tabel 4. 1. Tabel Spesifikasi Perangkat Lunak untuk Pengembangan Sistem

Fungsional	Teknologi	Gambar	Keterangan
Bahasa Pemrograman	Adobe Flash CC dengan Action Script 3.0		*Tools* pengembangan animasi dan *game* yang mendukung *Object Oriented Programming* dan mampu menghasilkan

Fungsional	Teknologi	Gambar	Keterangan
			produk dalam banyak bentuk
Script Editor	Flash Develop 4.0.4 RTM		*Editor* Script yang mampu mendukung banyak bahasa, salah satunya ActionScript 3.0
Sistem Enkripsi Internal I (*Anti Shared Object* [*saved data*] *Hacking*)	Sapphire SO Module		Untuk mencegah manipulasi data-data yang tersimpan dalam lokal komputer sehingga user tidak melakukan kecurangan
Sistem Enkripsi Internal II (*Anti Cheat Engine*)	MochiDigits		Untuk menghindari penggunaan *cheat engine* (program yang mampu merubah nilai-nilai dalam memori komputer)
Sistem Operasi	Windows		Pemain dapat memainkan *game* ini pada komputer dengan sistem operasi Windows

4.2. Implementasi Proses Modul *Game*

Pada bagian ini akan dijelaskan mengenai implementasi proses modul ke dalam sebuah kode. Implementasi proses aplikasi ini dilakukan dengan menggunakan bahasa pemrograman Action Script 3.0.

4.2.1. Implementasi Submodul Cellular Automata Map Generator

Pada subbab ini akan dijelaskan mengenai proses implementasi pembangkitan peta dengan *Cellular Automata*. Submodul ini memberikan keluaran *string* pembentuk peta. Keluaran tersebut kemudian menjadi masukan untuk Submodul Isometric Map Generator. Kelas-kelas yang digunakan untuk mengimplementasikan Submodul Cellular Automata Map Generator yaitu TileCA, dan CellularAutomata.

4.2.1.1. Kelas TileCA

Kelas Node merepresentasikan *tile* pada *Cellular Automata*. Kelas ini dapat dilihat pada kode sumber di bawah ini.

Kelas TileCA

```
public class TileCA extends MovieClip
{
    private var idTile:int = 0;
    public function setIdTile(_idTile:int = 0):void {
        idTile = _idTile;
        gotoAndStop(idTile);
    }

    public function getIdTile():int {
        return idTile;
    }

    private var tileWidth:Number = 0;
    public function setTileWidth(_tileWidth:Number =
    0):void {
```

```
      tileWidth = _tileWidth;
      this.height = tileWidth;
      this.width = tileWidth;
   }

   public function getTileWidth():Number {
      return tileWidth;
   }
}
```

Kode Sumber 4. 1 Kelas TileCA

4.2.1.2. Kelas CellularAutomata

Kelas ini berfungsi untuk membangkitkan peta permainan menggunakan *Cellular Automata*. Kelas ini dapat dilihat pada lampiran kode sumber 7.3.

4.2.2. Implementasi Submodul Isometric Map Generator

Pada subbab ini akan dijelaskan mengenai proses pengolahan *string* pembentuk peta dari Submodul Map Generator menjadi peta isometri. *String* dipecah terlebih dahulu dengan beberapa karakter pemisah atau *splitter* hingga membentuk *string* peta dengan format tertentu yang bisa diterjemahkan menjadi peta isometri. Terdapat dua buah kelas yang digunakan untuk mengimplementasikan Submodul Isometric Map Generator yaitu kelas Tile, dan TileManager. Implementasi dari masing-masing kelas tersebut adalah sebagai berikut.

4.2.2.1. Kelas Tile

Kelas Tile merupakan kelas satuan *tile* yang menyusun peta isometri. Tiap-tiap *tile* memiliki *id* yang merepresentasikan objek apa yang akan dipasang pada *tile* tersebut. Kelas ini dapat dilihat pada kode sumber di bawah ini.

```
Kelas Tile
public class Tile extends MovieClip
{
  private var idColour:int = 0;
```

```
  public function setIdColour(_idColour:int = 0,
  _updateDisplay:Boolean = false):void {
     idColour = _idColour;
     if (_updateDisplay) {
      updateDisplay();
     }
  }
  public function getIdColour():int {
     return idColour;
  }

  private var idColumn:int = 0;
  public function setIdColumn(_idColumn:int = 0):void {
     idColumn = _idColumn;
  }

  public function getIdColumn():int {
     return idColumn;
  }

  private var idRow:int = 0;
  public function setIdRow(_idRow:int = 0):void {
     idRow = _idRow;
  }
  public function getIdRow():int {
     return idRow;
  }
  private var tileWidth:Number = 0;
  public function setTileWidth(_tileWidth:Number =
  0):void {
     tileWidth = _tileWidth;
     width = getTileWidth();
     height = 0.5 * width;
  }
  public function getTileWidth():Number {
     return tileWidth;
  }
}
```

Kode Sumber 4. 2 Kelas Tile

4.2.2.2. Kelas TileManager

Kelas TileManager berfungsi untuk mengolah masukan *string* pembentuk peta ke dalam bentuk isometri. Implementasi kelas TileManager dapat dilihat pada lampiran kode sumber 7.1.

4.2.3. Implementasi Submodul Score Processor

Pada subbab ini akan dijelaskan mengenai proses pengolahan skor permainan. Kelas–kelas yang terlibat dalam submodul ini, yaitu kelas ScoreProcessor dan kelas EloRating. Implementasi dari masing-masing kelas tersebut adalah sebagai berikut.

4.2.3.1. Kelas ScoreProcessor

Kelas ScoreProcessor merupakan kelas yang menjadi antarmuka antara Submodul ScoreProcessor dengan submodul yang lain. Dalam kelas ini dilakukan operasi pengolahan hasil skor berdasarkan masukan dua variabel nilai, yatu sisa waktu *game*, dan jumlah musuh yang terbunuh. Implementasi kelas ini dapat dilihat pada kode sumber di bawah ini.

Kelas ScoreProcessor

```
public class ScoreProcessor extends MovieClip
{
   private var numberOfEnemyKilled:int = 0;
   public function
   setNumberOfEnemyKilled(_numberOfEnemyKilled:int =
   0):void {
        numberOfEnemyKilled = _numberOfEnemyKilled;
   }

   public function getNumberOfEnemyKilled():int {
        return numberOfEnemyKilled;
   }

   private var time:int = 0;
   public function setTime(_time:int = 0):void {
        time = _time;
   }
```

```
   public function getTime():int {
       return time;
   }

   public var levelScore:Number = 0;
   public static const constanta1:Number = 0.35;
   public static const constanta2:Number = 0.65;

   private function calculateScore ():void {
     levelScore = (level * 100);
     if
     (GamePlayDataManager.getInstance().getArrayOfEnemy()
     .length > 0) {
        for (var i:int = 0; i <
        GamePlayDataManager.getInstance().getArrayOfEnemy
        ().length; i++) {
           var idEnemy:int =
           GamePlayDataManager.getInstance().getEnemyByIn
           dex(i);
           levelScore +=

           DatabaseLevelScore.getInstance().getLevelScore
           (idEnemy);
         }
        }

      score = levelScore + (constanta1 *
      parseFloat(time_txt.text) + constanta2 *
      parseFloat(enemyKilled_txt.text));
   }
}
```

Kode Sumber 4. 3 Kelas ScoreProcessor

4.2.3.2. Kelas EloRating

Kelas EloRating merupakan kelas untuk mengolah skor menjadi *rating*. *Rating* berfungsi sebagai nilai modal untuk mendapatkan kombinasi musuh pada level selanjutnya. Implementasi kelas ini dapat dilihat pada kode sumber di bawah ini.

Kelas EloRating

```
public class EloRating extends MovieClip{
public function resetRating():void {
  newRating = 0;
}

  private var newRating:Number;
  private var coefficient:Number = 100;
  private var win:Number;
  private var winExpectation:Number;
  private var difference:Number = 0;
  public static const WIN:Number = 1;
  public static const LOSE:Number = 2;
  public static const DRAW:Number = 3;
  public function calculateRating(_oldRating:Number,
  _score:Number, _result:int):Number {
     difference = Math.abs(_oldRating - _score);

     if (_oldRating >= _score) {
        winExpectation =
        DatabaseEloScoringProbability.getInstance().getHi
        gherScoreProbability(difference);
     }else {
        winExpectation =
        DatabaseEloScoringProbability.getInstance().getLo
        werScoreProbality(difference);
     }

     switch(_result) {
        case WIN:
          win = 1;
          break;
        case LOSE:
          win = 0;
          break;
        case DRAW:
          win = 0.5;
          break;
     }
```

```
        newRating = _oldRating + coefficient * (win -
        winExpectation);

        return newRating;
    }
}
```

Kode Sumber 4. 4 Kelas EloRating

4.2.4. Implementasi Submodul Enemy Generator

Pada subbab ini akan dijelaskan mengenai proses implementasi pembangkitan anggota tim musuh. Submodul ini akan mengolah masukan berupa skor dari Submodul Score Processor. Terdapat dua buah kelas yang digunakan untuk mengimplementasikan Submodul Enemy Generator yaitu kelas Enemy dan CoinChangingProblem. Implementasi dari masing-masing kelas tersebut adalah sebagai berikut.

4.2.4.1. Kelas Enemy

Kelas Enemy berfungsi untuk merepresentasikan musuh. Kelas ini dapat dilihat pada kode sumber di bawah ini.

Kelas Enemy

```
public class Enemy extends MovieClip
{
    private var enemyGrade:int = 0;
    public function setEnemyGrade(_enemyGrade:int =
    0):void {
        enemyGrade = _enemyGrade;
    }

    public function getEnemyGrade():int {
        return enemyGrade;
    }

    private var idEnemy:int = 0;
    public function setIdEnemy(_idEnemy:int = 0):void
    {
        idEnemy = _idEnemy;
    }
```

```
    public function getIdEnemy():int {
        return idEnemy;
    }
}
```

Kode Sumber 4. 5 Kelas Enemy

4.2.4.2. Kelas CoinChangingProblem

Kelas ini berfungsi untuk membangkitkan kombinasi anggota tim musuh dengan menerapkan konsep *Coin Changing Problem* menggunakan algoritma *Dynamic Programming*. Kelas ini dapat dilihat pada lampiran kode sumber 7.2.

4.3. Implementasi Antarmuka

Pada subbab ini akan dijelaskan mengenai implementasi antarmuka *game* Pandawa:the Heroes of Ngastina. Terdapat tiga jenis antarmuka yang akan diimplementasikan, yaitu antarmuka *main menu game*, antarmuka *level menu game*, dan antarmuka tampilan profil musuh.

4.3.1. Implementasi Antarmuka *Main Menu*

Tampilan ini menunjukkan tampilan menu utama *game* Pandawa:the Heroes of Ngastina. *Main Menu* berisi menu awal permainan, yaitu menu “Mainkan”. Tampilan ini akan muncul setelah *file game* berhasil dimuat. Implementasi *main menu game* dapat dilihat pada Gambar 4.1.

Gambar 4. 1 Tampilan Antarmuka Main Menu

4.3.1.1. Implementasi Antarmuka Level Menu

Tampilan ini menunjukkan tampilan keseluruhan peta permainan yang juga merupakan level *game*. Terdapat sepuluh area sekaligus level yang akan dimainkan pemain. Implementasi *level menu game* dapat dilihat pada gambar di bawah ini.

Gambar 4. 2 Antarmuka Tampilan Level Menu

4.3.1.2. Implementasi Antarmuka Profil Musuh

Tampilan ini menunjukkan daftar musuh yang akan dihadapi pemain dalam *game*. Implementasi antarmuka profil musuh dapat dilihat pada gambar di bawah ini.

Gambar 4. 3 Antarmuka Tampilan Profil Musuh

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas tentang rangkaian pengujian dan evaluasi modul yang dilakukan dari hasil implementasi. Pengujian dilakukan untuk menguji secara keseluruhan apakah semua fungsionalitas berjalan sesuai keinginan. Pembahasan pada bab ini meliputi lingkungan pengujian, dasar pengujian, skenario pengujian, hasil pengujian, dan evaluasi.

5.1. Lingkungan Pelaksanaan Pengujian

Lingkungan pengujian merupakan komputer tempat pengujian modul dilakukan. Lingkungan pengujian ini menggunakan satu unit komputer. Spesifikasi lingkungan pengujian terbagi menjadi dua, yaitu lingkungan perangkat keras dan lingkungan perangkat lunak.

5.1.1. Lingkungan Perangkat Keras

Adapun spesifikasi perangkat keras yang digunakan pada pengujian ini adalah laptop Asus Ultrabook S400 dengan prosesor Intel Core i5, dan memori 4 GB.

5.1.2. Lingkungan perangkat lunak

Aplikasi ini dibangun dengan menggunakan Adobe Flash CC dan FlashDevelop 4.0. Untuk menjalankan aplikasi ini digunakan sistem operasi Windows 8.

5.2. Dasar Pengujian

Pengujian pada Modul Pembangkit Peta dan Level Dinamis ini menggunakan metode pengujian *black box* yang berfokus pada testing akurasi *rule* yang dihasilkan dan kebutuhan fungsional. Pengujian ini dilakukan untuk menguji apakah keputusan yang dihasilkan akurat dan fungsionalitas yang diidentifikasi pada tahap kebutuhan benar-benar diimplementasi dan bekerja seperti yang semestinya.

5.3. Skenario Pengujian

Pada subbab ini akan dijelaskan mengenai skenario pengujian dari modul Pembangkit Peta dan Level Dinamis pada *game* Pandawa:the Heroes of Ngastina.

.

5.3.1. Pengujian Fungsional

Berikut ini akan dijelaskan mengenai pengujian yang dilakukan pada Modul Pembangkit Peta dan Level Dinamis.

5.3.1.1. Pengujian Submodul Cellular Automata Map Generator

Pada bagian ini akan dijelaskan proses pengujian untuk Submodul Cellular Automata Map Generator. Pembentukan peta *Cellular Automata* dilakukan dengan tujuh kali iterasi. Terdapat tiga variabel pengujian dalam pembentukan peta ini, antara lain : *land ratio, obstacle coverage* (banyaknya halangan pada peta), ukuran *tile*, panjang peta, dan lebar peta.

a. Kasus Pengujian I

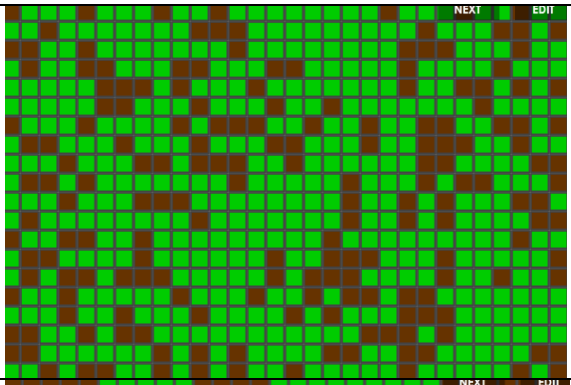
Pada kasus pengujian I, akan dibuat peta Cellular Automata dengan ukuran 30 x 20. Nilai-nilai pada tiap variabel untuk membentuk peta tersebut dapat dilihat pada Tabel 5.2.

Tabel 5. 1. Kasus Pengujian I pada Submodul Cellular Automata Map Generator

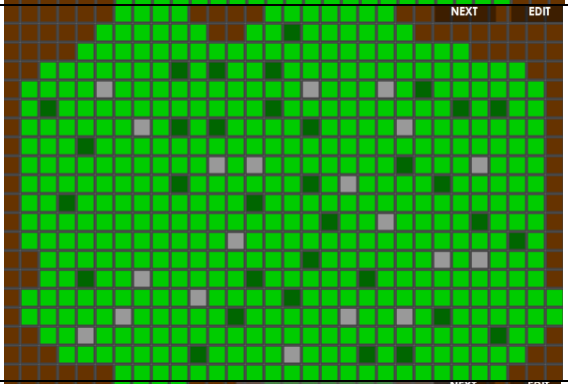
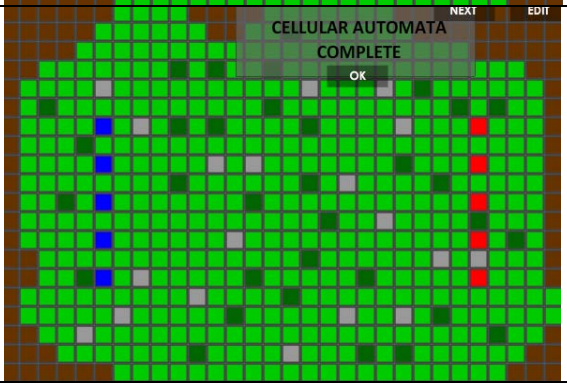
Kasus Pengujian I	
Variabel Pengujian	**Nilai**
Land Ratio	70% dari peta berupa daratan atau rumput
Obstacle Coverage	0.25
Ukuran *tile*	70
Panjang peta	30
Lebar peta	20

Proses pembentukan peta tiap iterasi menggunakan *Cellular Automata* sesuai dengan kasus pengujian I tersebut adalah sebagai berikut.

Tabel 5. 2 Langkah-Langkah Pembentukan Peta Cellular Automata Sesuai dengan Kasus Pengujian I

Tahap	**Hasil**
Inisialisasi	
Iterasi 1	

Tahap	Hasil
Iterasi 2	
Iterasi 3	
Iterasi 4	

Tahap	Hasil
Iterasi 5	
Iterasi 6	
Iterasi 7	

Dari peta *Cellular Automata* di atas, *string* pembentuk peta yang dihasilkan adalah sebagai berikut.

String Pembentuk Peta pada Kasus Pengujian I	
70#30#20>22222211112222111111122222222222221111112 21141111112222222222221111111111111111111122222221 11111141411411111111111112222111131111111131113141 11111221411111111111411111111141411221115131414111 14111131116111221114111111111111111111111112211115 11111313111111141116111221111111141111114131111411 1 11221141511111114111111111116111221111111111111114 11311114111221111511111311111111111161412221111111 11111141111113131112221145131111141111141111161112 21111111113111141111111111111121111131111141111131 1 31411111122113111111111111111111111141122221111114 1 11131114141111112222222211111111111111111111222	
Panjang *string*	609 karakter

b. Kasus Pengujian II

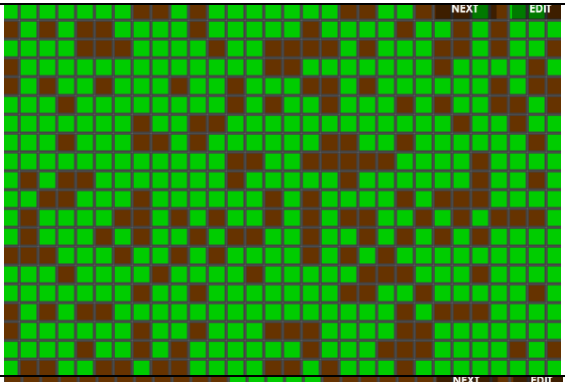
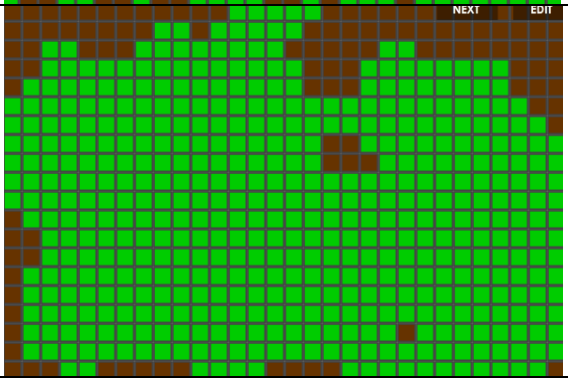
Pada kasus pengujian II, akan dibuat peta Cellular Automata dengan ukuran 30 x 20. Nilai-nilai pada tiap variabel untuk membentuk peta tersebut dapat dilihat pada Tabel 5.4.

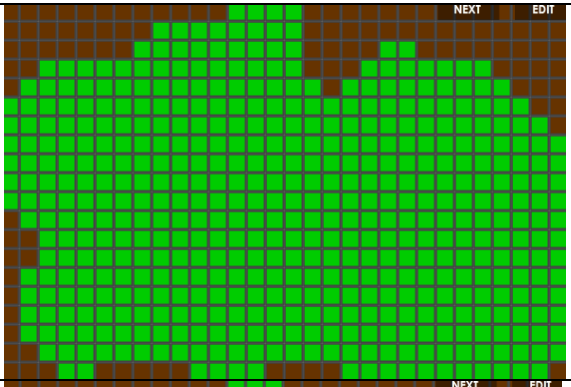
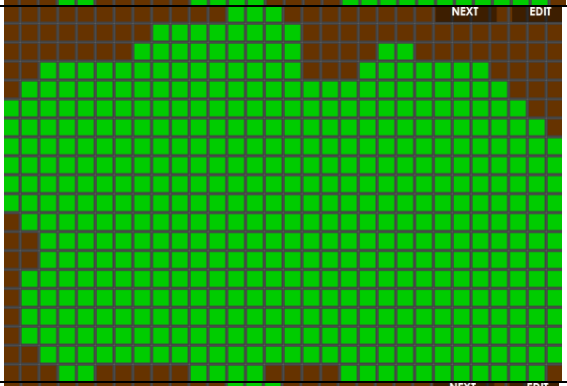
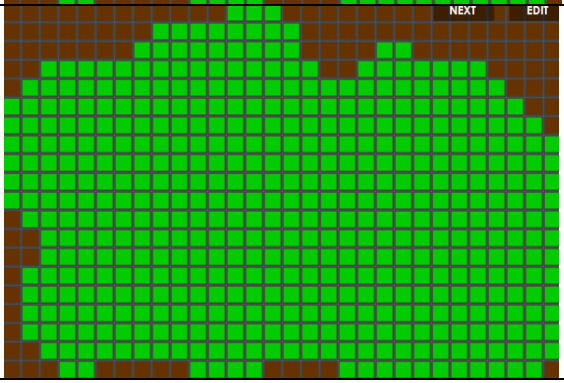
Tabel 5. 3 Kasus Pengujian II pada Submodul Cellular Automata Map Generator

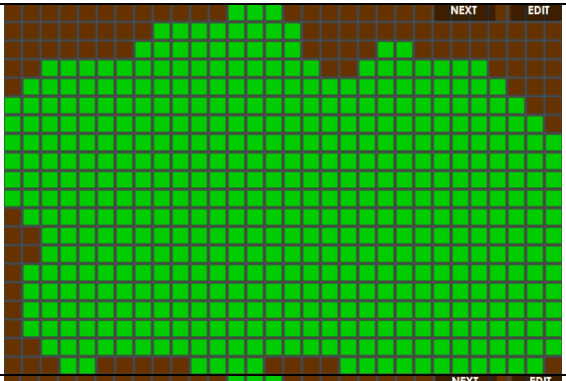
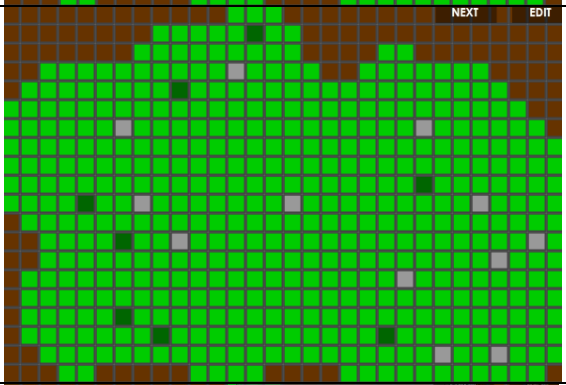
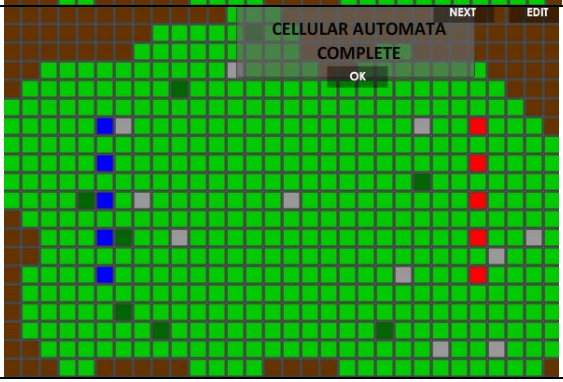
Kasus Pengujian II	
Variabel Pengujian	**Nilai**
Land Ratio	65% dari peta berupa daratan atau rumput
Obstacle Coverage	0.05
Ukuran *tile*	70
Panjang peta	30
Lebar peta	20

Proses pembentukan peta tiap iterasi menggunakan *Cellular Automata* sesuai dengan kasus pengujian II tersebut adalah sebagai berikut.

Tabel 5. 4 Langkah-Langkah Pembentukan Peta *Cellular Automata* Sesuai dengan Kasus Pengujian II

Tahap	**Hasil**
Inisialisasi	
Iterasi 1	

Iterasi 2	
Iterasi 3	
Iterasi 4	

Iterasi 5	
Iterasi 6	
Iterasi 7	

Dari peta *Cellular Automata* di atas, *string* pembentuk peta yang dihasilkan adalah sebagai berikut.

String Pembentuk Peta pada Kasus Pengujian II	
70#30#20>22222222222211122222222222222222222221111 14112222222222222222222211111111222211222222222221 11111111131111221111112222211111111411111111111111 11122211111111111111111111111111112211115311111111 11111131161112111111111111111111111111111111111115 11111111111111111161111111111111111111111111141111 11111145131111111311111111161111211111111111111111 11111111111122115413111111111111111161131221111111 11111111111111113111211151111111111111311161111 21111111111111111111111111112111114111111111111111 11111111211111141111111111411111111221111111111 11111111113113111222112222211112222111111111112	
Panjang *string*	609 karakter

5.3.1.2. Pengujian Submodul Isometric Map Generator

Pada bagian ini akan dijelaskan proses pengujian untuk Submodul Isometric Map Generator. *String* pembentuk peta yang dihasilkan dari pengujian Submodul Cellular Automata Map Generator akan menjadi masukan untuk submodul ini. *String* tersebut akan dipecah menjadi *array* 2 dimensi yang akan diolah menjadi peta isometri. Terdapat dua buah peta isometri yang akan dibangkitkan sesuai dengan dua buah pengujian pada subbab 5.3.1.1.

a. Pengujian Pembentukan Peta Isometri Sesuai Kasus Pengujian I

Masukan *string* pembentuk peta untuk kasus pengujian I adalah sebagai berikut.

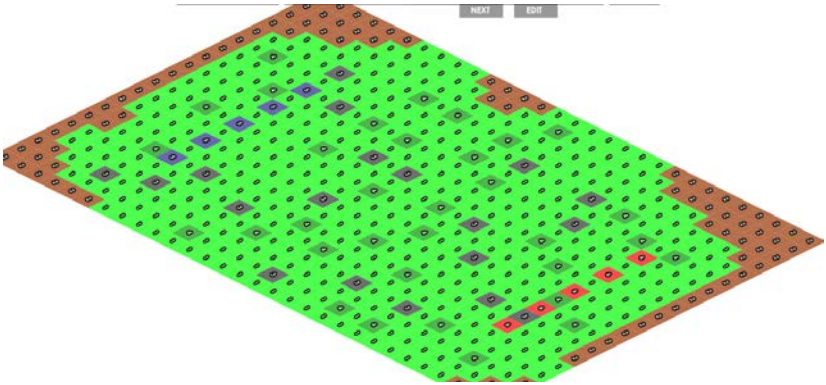
String Pembentuk Peta pada Kasus Pengujian I
70#30#20>222222111122221111112222222222222221111112
211411111222222222222111111111111111111122222221
111111414114111111111111222111131111111111131113141
1111122141111111111114111111111414112211115131414111
141111311161112211141111111111111111111111112211115
111113131111111411161112211111111411111141311114111
112211415111111141111111111161112211111111111111114
113111141112211115111111311111111111161412221111111
111111141111113131112221145131111141111141111161112
211111111131111411111111111112111113111114111111311
314111111221131111111111111111111141122221111111141
11131114141111112222222211111111111111111111222

String di atas akan dikonversi menjadi *array* 2 dimensi. *Array 2* dimensi peta yang dihasilkan dari *string* di atas adalah sebagai berikut.

```
[
[2,2,2,2,2,2,1,1,1,1,2,2,2,2,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2],
[2,2,2,2,2,1,1,1,1,1,1,2,2,1,1,4,1,1,1,1,1,1,2,2,2,2,2,2,2,2],
[2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2],
[2,2,1,1,1,1,1,1,1,4,1,4,1,1,4,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2],
[2,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,3,1,4,1,1,1,1,1,1,2],
[2,1,4,1,1,1,1,1,1,1,1,1,1,1,4,1,1,1,1,1,1,1,1,1,4,1,4,1,1,2],
[2,1,1,1,1,5,1,3,1,4,1,4,1,1,1,1,4,1,1,1,1,3,1,1,1,6,1,1,1,2],
[2,1,1,1,4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2],
[2,1,1,1,1,5,1,1,1,1,1,3,1,3,1,1,1,1,1,1,1,4,1,1,1,6,1,1,1,2],
[2,1,1,1,1,1,1,1,1,4,1,1,1,1,1,1,4,1,3,1,1,1,1,4,1,1,1,1,1,2],
[2,1,1,4,1,5,1,1,1,1,1,1,1,4,1,1,1,1,1,1,1,1,1,1,1,6,1,1,1,2],
[2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,4,1,1,3,1,1,1,1,4,1,1,1,2],
```

```
[2,1,1,1,1,5,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,6,1,4,1,2],
[2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,4,1,1,1,1,1,1,3,1,3,1,1,1,2],
[2,2,1,1,4,5,1,3,1,1,1,1,1,4,1,1,1,1,1,4,1,1,1,1,1,6,1,1,1,2],
[2,1,1,1,1,1,1,1,1,1,3,1,1,1,1,4,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[2,1,1,1,1,1,3,1,1,1,1,1,1,4,1,1,1,1,1,3,1,1,3,1,4,1,1,1,1,1,1],
[2,2,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,4,1,1,2],
[2,2,2,1,1,1,1,1,1,1,4,1,1,1,1,3,1,1,1,4,1,4,1,1,1,1,1,1,2,2],
[2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2]
]
```

Peta isometri yang dihasilkan dari *array* 2 dimensi di atas dapat dilihat pada Gambar 5.1. Luas peta isometri yang dihasilkan pada kasus pengujian I adalah 2.940.000 *pixel*.



Gambar 5. 1. Hasil Pengolahan dari Keluaran Submodul Map Generator Menjadi Peta Isometri pada Kasus Pengujian I

b. Pengujian Pembentukan Peta Isometri Sesuai Kasus Pengujian II

Masukan *string* pembentuk peta untuk kasus pengujian II adalah sebagai berikut.

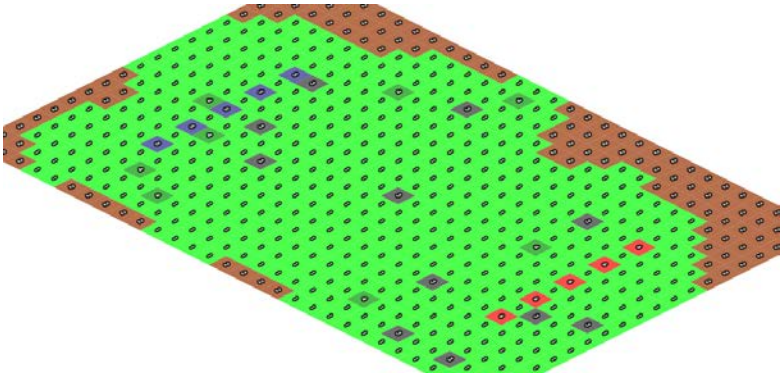
String Pembentuk Peta pada Kasus Pengujian II
70#30#20>22222222222211122222222222222222222221111
14112222222222222222222221111111122221122222222221
11111111311112211111122222111111114111111111111111
11122211111111111111111111111111122111115311111111
11111131161112111111111111111111111111111111111115
11111111111111111161111111111111111111111111141111
11111145131111111311111111161111211111111111111111
11111111111221115411311111111111111161131221111111
11111111111111113111211115111111111111111311161111
21111111111111111111111111112111114111111111111111
11111111211111141111111111141111111112211111111111
11111111113113111222112222211112222111111111112

String di atas akan dikonversi menjadi *array* 2 dimensi. *Array 2* dimensi peta yang dihasilkan dari *string* di atas adalah sebagai berikut.

```
[
[2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],
[2,2,2,2,2,2,2,2,2,1,1,1,1,1,4,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2],
[2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,2,2,2,2,2,1,1,2,2,2,2,2,2,2,2],
[2,2,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,2,2,1,1,1,1,1,1,1,2,2,2,2,2],
[2,1,1,1,1,1,1,1,1,1,4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2],
[1,1,1,1,1,1,5,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,6,1,1,1,2],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[1,1,1,1,1,1,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,6,1,1,1,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,4,1,1,1,1,1,1,1],
[1,1,1,1,4,5,1,3,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,1,1,6,1,1,1,1],
```

```
[2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[2,2,1,1,1,5,4,1,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,6,1,1,3,1],
[2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1],
[2,1,1,1,1,5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,6,1,1,1,1],
[2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[2,1,1,1,1,1,4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[2,1,1,1,1,1,1,1,4,1,1,1,1,1,1,1,1,1,1,1,4,1,1,1,1,1,1,1,1,1],
[2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,3,1,1,1],
[2,2,2,1,1,2,2,2,2,2,1,1,1,1,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,2]
]
```

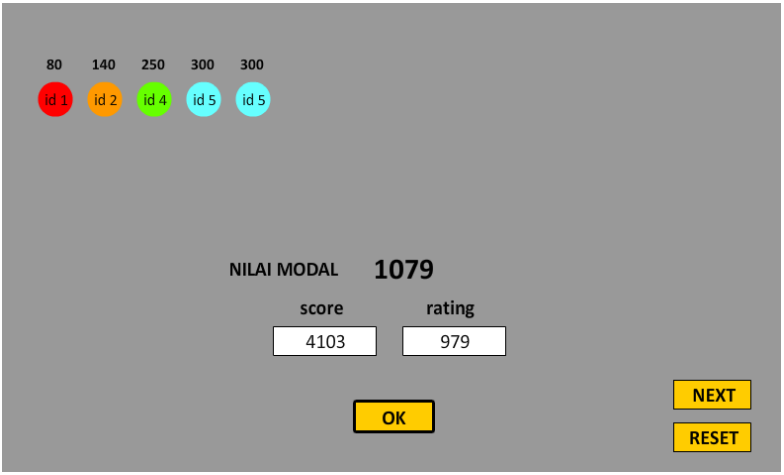
Peta isometri yang dihasilkan dari *array* 2 dimensi di atas dapat dilihat pada Gambar 5.2. Luas peta isometri yang dihasilkan pada kasus pengujian II adalah 2.940.000 *pixel*.



Gambar 5. 2 Hasil Pengolahan dari Keluaran Submodul Map Generator Menjadi Peta Isometri pada Kasus Pengujian II

5.3.1.3.Pengujian Submodul Enemy Generator dan Submodul Score Processor

Pada bagian ini akan dijelaskan proses pengujian untuk Submodul Enemy Generator dan Score Processor. Pembangkitan anggota tim musuh memerlukan masukan dua buah nilai yaitu waktu yang tersisa dan jumlah musuh yang terbunuh.



Gambar 5. 3 Hasil Skor Pemain dan Pembangkitan Tim Musuh

Di bawah ini adalah pengujian pembangkitan anggota tim musuh dengan dua jenis kasus uji. Kasus uji pertama adalah pemain dapat menyelesaikan *game* dengan masih memiliki sisa waktu. Kasus uji kedua, pemain dapat menyelesaikan permainan namun tidak mempunyai waktu sisa permainan atau sisa waktu permainannya adalah nol.

Pengujian Pertama

Kasus Pengujian : pemain menyelesaikan *game* dengan masih memiliki sisa waktu permainan.

Tabel 5. 5 Tabel Hasil Pembangkitan Anggota Tim Musuh Berdasarkan Pengujian Pertama

Lv	Nilai Modal	Kombinasi Musuh	Sisa Waktu	*Score*	*Rating* pemain
1	160	Sengkuni (80), Sengkuni (80)	103 detik	387	239
2	239	Sengkuni (80), Dursasana (140)	200 detik	696	334
3	334	Sengkuni (80), Duryudana (250)	180 detik	989	433
4	433	Sengkuni (80), Dursasana (140), Karna (200)	120 detik	1468	533
5	533	Sengkuni (80), Karna (200), Duryudana (250)	72 detik	1802	633
6	633	Sengkuni (80), Duryudana (250), Bogadenta (300)	110 detik	2165	733
7	733	Sengkuni (80), Karna (200), Karna (200), Duryudana (250)	120 detik	2869	833
8	833	Sengkuni (80), Karna (200), Duryudana (250), Bogadenta (300)	98 detik	3261	933
9	933	Sengkuni (80), Duryudana (250), Bogadenta (300), Bogadenta (300)	50 detik	3645	1033
10	1033	Sengkuni (80), Karna (200), Karna (200), Duryudana (250), Bogadenta (300)	25 detik	4537	-

Pengujian Kedua

Kasus Pengujian : pemain menyelesaikan *game* tanpa sisa waktu atau sisa waktu permainan adalah nol.

Tabel 5. 6**.** Tabel Hasil Pembangkitan Anggota Tim Musuh Berdasarkan Pengujian Kedua

Lv	Nilai Modal	Kombinasi Musuh	Sisa Waktu	*Score*	*Rating* pemain
1	160	Sengkuni (80), Sengkuni (80)	0 detik	351	235
2	235	Sengkuni (80), Dursasana (140)	0 detik	626	326
3	326	Sengkuni (80), Sengkuni (80), Sengkuni (80), Sengkuni (80)	0 detik	1252	426
4	426	Sengkuni (80), Dursasana (140), Karna (200)	0 detik	1426	526
5	526	Sengkuni (80), Dursasana (140), Bogadenta (300)	0 detik	1776	626
6	626	Sengkuni (80), Dursasana (140), Karna (200), Karna (200)	0 detik	2427	726
7	726	Sengkuni (80), Dursasana (140), Karna (200), Bogadenta (300)	0 detik	2827	826
8	826	Sengkuni (80), Dursasana (140), Bogadenta (300), Bogadenta (300)	0 detik	3227	926

Lv	Nilai Modal	Kombinasi Musuh	Sisa Waktu	*Score*	*Rating* pemain
9	926	Sengkuni (80), Dursasana (140), Karna (200), Karna (200), Bogadenta (300)	0 detik	4078	1026
10	1026	Sengkuni (80), Dursasana (140), Karna (200), Bogadenta (300), Bogadenta (300)	0 detik	4528	-

BAB VI
PENUTUP

Bab ini membahas kesimpulan akhir dan saran yang diperoleh selama proses pembuatan Modul Pembangkit Peta dan Level Dinamis pada *game* Pandawa:the Heroes of Ngastina.

6.1. Kesimpulan

Dalam proses pengerjaan Tugas Akhir dari tahap analisis, desain, implementasi, hingga pengujian didapatkan kesimpulan sebagai berikut:

1. Rasio perbandingan area, *rule*, dan jumlah iterasi pada proses pembentukan peta *Cellular Automata* sangat berpengaruh terhadap peta wilayah pertarungan yang dihasilkan.
2. *Rating* pemain berfungsi sebagai nilai modal yang akan ditukar dengan kombinasi musuh pada level selanjutnya dengan mengadaptasi *Coin-Changing Problem* menggunakan *Dynamic Programming*.
3. Skor pemain mempengaruhi *rating* pemain yang diolah menggunakan metode *Elo Rating System*. *Rating* menunjukkan kemampuan pemain pada suatu level, sehingga dari *rating* tersebut akan dihasilkan kombinasi musuh yang sesuai dengan kemampuan pemain.

6.2. Saran

Setelah pengerjaan Tugas Akhir ini selesai, terdapat beberapa saran yang bertujuan untuk menyempurnakan Modul Pembangkit Peta dan Level Dinamis ini. Saran untuk pengembangan modul ini antara lain :

1. Pembangkitan peta dan level dinamis dapat dilakukan pada *game* dengan *genre Real-Time Strategy (RTS)* dimana dalam *game* tersebut peta dan musuh dibangkitkan secara *real time.*

DAFTAR PUSTAKA

[1] Senocular. Isometric Perspective. *Kirupa.com.* [Online] September 20, 2014. http://www.kirupa.com/developer/isometric/perspective.htm.

[2] Bose, Juwal. Creating Isometric Worlds : A Primer for Game Developers. *Game Development Tutsplus.* [Online] May 27, 2013. http://gamedevelopment.tutsplus.com/tutorials/creating-isometric-worlds-a-primer-for-game-developers--gamedev-6511.

[3] *Procedural Content Generation for Games : A Survey.* Hendrikx, M. 2011, Games and Digital Entertainment (SBGAMES).

[4] *Dynamic Programming Solution to the Coin Changing Problem.* Aslam, Prof.

[5] *Dynamic Difficulty Adjustment in Computer Games.* Chang, David Michael Jordan.

[6] Cai, Xuan. *Canonical Coin System for Change-Making Problems.* Shanghai, China : Department of Computer Science and Engineering, Shanghai Jiao Tong University, 2009.

[7] Cormen, Thomas H., et al. *Introduction to Algorithm 3rd Edition.* Massachussetts : The MIT Press, 2009.

[8] Levitin, Anany. *Introduction to Design and Analysis of Algorithm.* New Jersey, USA : Pearson Education, Inc., 2012.

[9] Rogers, Scott. *Level Up! Guide to Great Video Game Design.* West Sussex, United Kingdom : A John Wiley and Son, Ltd, Publication, 2010.

[10] *A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft.* Ontanon, S. 2013, Computational Intelligence and AI in Games, IEEE Transaction.

[11] Schell, Jesse. *The Art of Game Design.* Burlington, MA, USA : Morgan Kauffman Publishers, 2008.

[12] Togelius, Julian and Schmidhudber, Jurgen. *An Experiment in Automatic Game Design.*
[13] Bester, D.W. and von Maltitz, M.J. *Introducing Momentum to Elo Rating System.* s.l. : University of the Free State, Department of Mathematical Statistics and Actuarial Science.
[14] Karri, Jarko. *Cellular Automata : Tutorial.* s.l. : Department of Mathematics, University of Turku Finland.
[15] Incorporated, Adobe System. *Programming ActionScript 3.0.* California, USA : s.n., 2007.
[16] Johnson, Soren. Analysis: Turn-Based Versus Real-Time . *Gamasutra.* [Online] Spore, Civilization IV, Agustus 2009. [Cited: Juli 8, 2015.] http://www.gamasutra.com/php-bin/news_index.php?story=25920.
[17] Barton, Matt and Loguidice, Bill. The History of Rogue: Have @ You, You Deadly Zs. *Gamasutra.* [Online] May 5, 2009. [Cited: Juli 8, 2015.] http://www.gamasutra.com/view/feature/4013/the_history_of_rogue_have__you_.php#comment19111.

LAMPIRAN A – KODE SUMBER

Kode Sumber Submodul Isometric Map Generator

Kode Sumber 7. 1 Kelas TileManager.as

Kelas TileManager.as

```
public class TileManager extends MovieClip
{
private function
initAllTiles(_latestIndexOfPreviousContent:int = -
1):void {
var tileHeight:Number = getTileWidth() / 2;
for (var i:int = 1; i <= getArrayHeight(); i++) {
for (var j:int = 1; j <= getArrayWidth(); j++) {
var idColour:int = 1;
try { idColour = getTileWithIdRowAndIdColumn(i,
j).getIdColour(); } catch (error:Error) {}
attachTile(getInitXStart() + (j - i) * getTileWidth() /
2, getInitYStart() + ((j + i) / 2 - 1) * tileHeight,
getTileWidth(), i, j);
getTile(arrayOfTiles.length - 1).setIdColour(idColour);
getTile(arrayOfTiles.length - 1).updateDisplay();
}
}

        removeTilesFromIndex0To(_latestIndexOfPreviousCon
tent != -1?_latestIndexOfPreviousContent:0);
               }      }
```

```
private var initYStart:Number = 0;
public function setInitYStart(_initYStart:Number =
0):void {
	initYStart = _initYStart;
}

public function getInitYStart():Number {
	return initYStart;
}

private var initXStart:Number = 0;
public function setInitXStart(_initXStart:Number =
0):void {
	initXStart = _initXStart;
}
public function getInitXStart():Number {
	return initXStart;
}

private var kindOfColour:int = 0;
public function setKindOfColour(_kindOfColour:int =
0):void {
	kindOfColour = _kindOfColour;
}

public function getKindOfColour():int {
	return kindOfColour;
}

private var arrayWidth:int = 0;
public function setArrayWidth(_arrayWidth:int = 0):void
{
	arrayWidth = _arrayWidth;
}

public function getArrayWidth():int {
	return arrayWidth;
}

private var arrayHeight:int = 0;
```

```
public function setArrayHeight(_arrayHeight:int =
0):void {
	arrayHeight = _arrayHeight;
}

public function getArrayHeight():int {
	return arrayHeight;
}

private var tileWidth:Number = 0;
public function setTileWidth(_tileWidth:Number = 0):void
{
	tileWidth = _tileWidth;
}

public function getTileWidth():Number {
return tileWidth;
}

var arrayOfSplit1:Array = _stringCode.split(">");
                        var arrayOfSplit2:Array =
(arrayOfSplit1[0] as String).split("#"); //main
properties
                        var arrayOfSplit3:Array =
(arrayOfSplit1[1] as String).split(""); //all tile
                        trace("arrayOfSplit3",
arrayOfSplit3);

                        this.x = 0;
                        this.y = 0;
                        setInitXStart(0);
                        setInitYStart(0);

	implementChange(parseFloat(arrayOfSplit2[2] as
String), parseFloat(arrayOfSplit2[1] as String), 50, 6);

                        var index:int = 0;
                        var arrayOfRow:Array;
                        var arrayOfIsoMap:Array = new
Array();
                        for (var i:int = 1; i <=
arrayOfSplit2[2]; i++) {
```

```
                                arrayOfRow = new Array();
                                for (var j:int = 1; j <=
arrayOfSplit2[1]; j++) {

        getTileWithIdRowAndIdColumn(i,
j).setIdColour(parseInt(arrayOfSplit3[index]), true);

        arrayOfRow.push(parseInt(arrayOfSplit3[index]));
                                        index++;
                                }

        arrayOfIsoMap.push(arrayOfRow);
}

public function implementChange(_arrayHeight:int,
_arrayWidth:int, _tileWidth:int,
_kindsOfColour:int):void {
        setArrayHeight(_arrayHeight);
        setArrayWidth(_arrayWidth);
        setTileWidth(_tileWidth);
        setKindOfColour(_kindsOfColour);
        initAllTiles(arrayOfTiles.length);
    }
}
```

Kode Sumber 7. 2. Kelas CoinChangingProblem.as

Kelas CoinChangingProblem.as

```
public class CoinChangingProblem extends MovieClip
{

private static var
coinChangingProblem:CoinChangingProblem = new
CoinChangingProblem();

public static function getInstance():CoinChangingProblem
{
        return coinChangingProblem;
}

private function initArray(_score:int):void {
        arrayOfC = new Array();
        arrayOfScore = new Array();
```

```
        arrayOfUsedCoin = new Array();
        arrayOfEnemyCombination = new Array();
        arrayOfEnemyCombinationIndex = new Array();
        arrayOfDenomination =
DatabaseEnemy.getInstance().getArrayOfEnemy();
        var lineOfC:Array;
        var lineOfUsed:Array;
        for (var i:int = 0; i <=
arrayOfDenomination.length; i++) {
                lineOfC = new Array();
                lineOfUsed = new Array();
                for (var j:int = 0; j <= _score; j++) {
                        lineOfC.push(0);
                        lineOfUsed.push(0);
                }
                arrayOfC.push(lineOfC);

        arrayOfUsedCoin.push(lineOfUsed);
}

        for (var k:int = 0; k <= _score; k++) {
                arrayOfScore.push(k);
        }
}

private var arrayOfEnemyCombination:Array;
private var arrayOfEnemyCombinationIndex:Array;
public static const GENERATE_COMBINATION:String =
"GENERATE_COMBINATION";
private function optimalCoinSet(_i:int, _j:int,
_arrayOfDenomination:Array,
_arrayOfCoinUsed:Array):Array {
        if (_j == 0) {
                return [0];
        }

        if (_i == 6) {
        arrayOfEnemyCombination.push(arrayOfEnemyCombinat
ion[1]);
        arrayOfEnemyCombinationIndex.push(1);
        arrayOfEnemyCombination.push(arrayOfEnemyCombinat
ion[1]);
```

```
        arrayOfEnemyCombinationIndex.push(1);
                return arrayOfEnemyCombination;
        }
        if (_arrayOfCoinUsed[_i][_j]) {
        arrayOfEnemyCombination.push(_arrayOfDenomination
[_i]);
        arrayOfEnemyCombinationIndex.push(_i);
        optimalCoinSet(_i, _j - _arrayOfDenomination[_i],
_arrayOfDenomination, _arrayOfCoinUsed);
        dispatchEvent(new MGEvent(GENERATE_COMBINATION,
true, false, arrayOfEnemyCombination));
        return arrayOfEnemyCombination;
        }else {
        optimalCoinSet(_i + 1, _j, _arrayOfDenomination,
_arrayOfCoinUsed);
        dispatchEvent(new MGEvent(GENERATE_COMBINATION,
true, false, arrayOfEnemyCombination));
        return arrayOfEnemyCombination;
                        }
                }

                private var arrayOfDenomination:Array =
new Array();
                private var arrayOfScore:Array = new
Array();
                private var arrayOfUsedCoin:Array = new
Array();
                private var arrayOfC:Array = new Array();
                private var score:int = 0;
                public function changeCoin(_score:int =
0):void {
                                initArray(_score);
                                score = _score;
                                var n:int =
arrayOfDenomination.length;
                                for (var j:int = 0; j <=
_score; j++) {
                                        arrayOfC[n][j] = j;

        arrayOfUsedCoin[n][j] = true;
                                }
```

```
        for (var i:int = n - 1; i >= 0; i--) {
                                        for (var k:int = 0;
k <= _score; k++) {
                                                if
(arrayOfDenomination[i] > arrayOfScore[k] || arrayOfC[i
+ 1][arrayOfScore[k]] < 1 + arrayOfC[i][arrayOfScore[k]
- arrayOfDenomination[i]]) {

        arrayOfC[i][arrayOfScore[k]] = arrayOfC[i +
1][arrayOfScore[k]];

        arrayOfUsedCoin[i][arrayOfScore[k]] = false;
                                                }else {

        arrayOfC[i][arrayOfScore[k]] = 1 +
arrayOfC[i][arrayOfScore[k] - arrayOfDenomination[i]];

        arrayOfUsedCoin[i][arrayOfScore[k]] = true;
                                                }
                                        }
                                }
                                for (var l:int = 0; l <
arrayOfDenomination.length; l++) {
                                //      trace("l", l,
arrayOfC[l]);
                                }

        for (var m:int = 0; m <
arrayOfDenomination.length; m++) {

        }
                        if (arrayOfEnemyCombination.length
< 2) {

        arrayOfEnemyCombination.push(arrayOfEnemyCombinat
ion[1]);

        arrayOfEnemyCombinationIndex.push(1);

        arrayOfEnemyCombination.push(arrayOfEnemyCombinat
ion[1]);
```

```
        arrayOfEnemyCombinationIndex.push(1);
                        }
                }
}
```

Kode Sumber 7. 3. Kelas CellularAutomata.as

Kelas CellularAutomata.as

```
package mapgenerator
{
        import caurina.transitions.Tweener;
        import com.greensock.TweenMax;
        import flash.display.MovieClip;
        import flash.events.Event;
        import flash.events.MouseEvent;
        import flash.net.FileReference;

        public class CellularAutomata extends MovieClip
        {

                private static const tileWidth:int = 20;
                private static const TILE_WIDTH:int = 70;
                private static const MAP_WIDTH:int = 42;
        private static const MAP_HEIGHT:int = 28;

        private static const ID_TILE_LAND:int = 1;
        private static const ID_TILE_FOREST:int = 2;
        private static const ID_TILE_STONE:int = 3;
        private static const ID_TILE_TREE:int = 4;
        private static const ID_TILE_PLAYER:int = 5;
        private static const ID_TILE_ENEMY:int = 6;

        private var initLandRatio:Number = 0.7;
        private var array2DOfNodesMap:Array = new
Array();

        private static const PLAYERS_TEMPLE:int = 1;
        private static const ENEMIES_TEMPLE:int = 2;

        private static const PLAYERS:int = 1;
        private static const ENEMIES:int = 2;
```

```
	public function CellularAutomata()
	{
		super();
		addEventListener(Event.ADDED_TO_STAGE,
init);
	}

	private function init(_event:Event = null):void {

	removeEventListener(Event.ADDED_TO_STAGE, init);
			attachAllNodes();
			onFrequentlyDoProcess();
		}

private function onFrequentlyDoProcess():void {
			if (counterProcess <= 8) {
				doProcess();
			}
			TweenMax.delayedCall(1,
onFrequentlyDoProcess);
		}
		private var arrayOf2DMap:Array = new
Array();
		private function attachAllNodes():void {
			for (var i:int = 1; i <=
MAP_HEIGHT; i++) {
				var
arrayOfSingleRowInMap:Array = new Array();
				var arrayOfSingleRow:Array
= new Array();
				for (var j:int = 1; j <=
MAP_WIDTH; j++) {

	attachNode(tileWidth);

	getNode(arrayOfNodes.length - 1).x = (j - 1) *
tileWidth;

	getNode(arrayOfNodes.length - 1).y = (i - 1) *
tileWidth;
					var idTile:int;
```

```
                                        if (Math.random() <
initLandRatio) {
                                                idTile =
ID_TILE_LAND;
                                        }else {
                                                idTile =
ID_TILE_FOREST;
                                        }

        getNode(arrayOfNodes.length -
1).setIdTile(idTile);

        arrayOfSingleRowInMap.push(getNode(arrayOfNodes.l
ength - 1));

        arrayOfSingleRow.push(idTile);
                                }

        array2DOfNodesMap.push(arrayOfSingleRowInMap);

        arrayOf2DMap.push(arrayOfSingleRow);
                        }
                }
                private var counterProcess:int = 0;
                private var outputString:String;
                private var encodedString:String;
                private var customization:String = "";
                private function doProcess():void {
                        for (var i:int = 0; i <
MAP_HEIGHT; i++) {
                                for (var j:int = 0; j <
MAP_WIDTH; j++) {
                                        if (counterProcess
<= 4) {

        createAndUpdateCellularAutomata(i, j);
                                        } else if
(counterProcess == 5) {
                                                if
(numberOfObstacles < 1) {

        placeTreeAndStone();
```

```
        numberOfObstacles++;
                                }
                } else if (counterProcess == 6) {

        placePlayersAndEnemies(PLAYERS);
                placePlayersAndEnemies(ENEMIES);
        }
                }
        }
        counterProcess++;
        if (counterProcess == 7) {
                onAllProcessesDone();
        }
}
                private function onAllProcessesDone():void
{

        TweenMax.killDelayedCallsTo(onFrequentlyDoProcess
);
                        outputString = TILE_WIDTH + "#" +
MAP_WIDTH + "#" + MAP_HEIGHT + ">";
                        customization = outputString;
                        for (var k:int = 0; k <
MAP_HEIGHT; k++) {
                                for (var l:int = 0; l <
MAP_WIDTH; l++) {

                        outputString +=
((array2DOfNodesMap[k][l] as Node).getIdTile());
                                        arrayOf2DMap[k][l] =
(array2DOfNodesMap[k][l] as Node).getIdTile();
                                        trace(k, l,
(array2DOfNodesMap[k][l] as Node).getIdTile());
                                }
        }
}

private function createAndUpdateCellularAutomata(i:int,
j:int):void {
                        //the making of land and forest
                        var numberOfGreenAround:int = 0;
```

```
                        try { if ((array2DOfNodesMap[i -
1][j - 1] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i
][j - 1] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i +
1][j - 1] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i +
1][j    ] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i +
1][j + 1] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i
][j + 1] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i -
1][j + 1] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        try { if ((array2DOfNodesMap[i -
1][j    ] as Node).currentFrame == ID_TILE_LAND) {
numberOfGreenAround++; } } catch (error:Error) {}
                        if ((array2DOfNodesMap[i][j] as
Node).currentFrame == ID_TILE_FOREST) { //FOREST
                                if (numberOfGreenAround >=
5) {

       (array2DOfNodesMap[i][j] as
Node).gotoAndStop(ID_TILE_LAND);

       (array2DOfNodesMap[i][j] as
Node).setIdTile(ID_TILE_LAND);
                                }
                        } else if
((array2DOfNodesMap[i][j] as Node).currentFrame ==
ID_TILE_LAND) { //LAND
                                if (numberOfGreenAround >=
4) {
```

```
       (array2DOfNodesMap[i][j] as
Node).gotoAndStop(ID_TILE_LAND);

       (array2DOfNodesMap[i][j] as
Node).setIdTile(ID_TILE_LAND);
                              } else {

       (array2DOfNodesMap[i][j] as
Node).gotoAndStop(ID_TILE_FOREST);

       (array2DOfNodesMap[i][j] as
Node).setIdTile(ID_TILE_FOREST);
                              }
                       }
               }

private var playerTempleCreated:Boolean = false;
private var enemyTempleCreated:Boolean = false;
private var grid:Array = new Array();

private var numberOfObstacles:int = 0;
private var obstacleCoverage:Number = 0.05;
private function placeTreeAndStone():void {
       for (var i:int = 0; i < MAP_HEIGHT; i++) {
              for (var j:int = 0; j < MAP_WIDTH; j++) {
                     var random:Number = Math.random();
                     if (random < obstacleCoverage) {
                            try {
                                   if
((array2DOfNodesMap[i    ][j    ] as Node).getIdTile()
== ID_TILE_LAND &&
       (array2DOfNodesMap[i - 1][j - 1] as
Node).getIdTile() == ID_TILE_LAND &&
       (array2DOfNodesMap[i    ][j - 1] as
Node).getIdTile() == ID_TILE_LAND &&
       (array2DOfNodesMap[i + 1][j - 1] as
Node).getIdTile() == ID_TILE_LAND &&
       (array2DOf[NodesMap[i + 1][j    ] as
Node).getIdTile() == ID_TILE_LAND &&
       (array2DOfNodesMap[i + 1][j + 1] as
Node).getIdTile() == ID_TILE_LAND &&
```

```
       (array2DOfNodesMap[i    ][j + 1] as
Node).getIdTile() == ID_TILE_LAND &&
       (array2DOfNodesMap[i - 1][j + 1] as
Node).getIdTile() == ID_TILE_LAND &&
       (array2DOfNodesMap[i - 1][j    ] as
Node).getIdTile() == ID_TILE_LAND) {

               var idTile:int = Math.floor(Math.random()
* 2) + 3;

               (array2DOfNodesMap[i][j] as
Node).setIdTile(idTile);

       }
       }catch (error:Error) { }
                                       }
                                }
                        }
               }
               private function
checkTileToPlaceCharacter():void {

               }
               private var numberOfPlayers:Number = 5;
               private var numberOfEnemies:Number = 5;
               private function
placePlayersAndEnemies(_typeOfCharacter:int):void {
                       var numberOfTotalCharacter:Number
= 0;
                       var idTile:int;

                       var row:int;
                       var rowChosen:int;
                       var numberOfRow:int =
arrayOfNodes.length;
                       var col:int;
                       var colChosen:int;

                       if (_typeOfCharacter == PLAYERS) {
                              col = 0 + 10;
                              row = MAP_HEIGHT / 2;
```

```
                                numberOfTotalCharacter =
numberOfPlayers;
                                idTile = ID_TILE_PLAYER;
                                for (var i:int = 1; i <= 5;
i++) {
                                        rowChosen = row + 2
* (i - Math.ceil(numberOfTotalCharacter / 2));
                                        colChosen = col;

       (array2DOfNodesMap[rowChosen][colChosen] as
Node).setIdTile(idTile);
                                }
                        }else if (_typeOfCharacter ==
ENEMIES) {
                                col = MAP_WIDTH - 10;
                                row = MAP_HEIGHT / 2;
                                numberOfTotalCharacter =
numberOfEnemies;
                                idTile = ID_TILE_ENEMY;
                                for (var i:int = 1; i <= 5;
i++) {
                                        rowChosen = row + 2
* (i - Math.ceil(numberOfTotalCharacter / 2));
                                        colChosen = col;

       (array2DOfNodesMap[rowChosen][colChosen] as
Node).setIdTile(idTile);
                                }
                        }
                }
                //}endregion INITIALIZATION


                //{subregion Node
                private var arrayOfNodes:Array = new
Array();
                private function
attachNode(_tileWidth:Number):void {
                        var node:Node = new
Node(_tileWidth);
```

```
		node.addEventListener(Node.EVENT_REMOVE,
removeNode);
				arrayOfNodes.push(node);
				addChild(node);
			}
			private function getNode(_idOnItsArray:int
= 0):Node {
				return
(arrayOfNodes[_idOnItsArray] as Node);
			}
			private function getNumberOfNode():int {
				return arrayOfNodes.length;
			}
			private function removeAllNodes():void {
				while (arrayOfNodes.length > 0) {

		(arrayOfNodes[arrayOfNodes.length - 1] as
Node).remove();
				}
			}
			private function removeNode(_event:Event =
null):void {
				(_event.currentTarget as
Node).removeEventListener(Node.EVENT_REMOVE,
removeNode);
				removeChild(_event.currentTarget
as Node);
		arrayOfNodes.splice(arrayOfNodes.indexOf(_event.c
urrentTarget as Node), 1);
			}
			//}endsubregion Node

			private function
removeAllExistingObjects():void {
				removeAllPopUp_Dones();
				removeAllNodes();
			}
			//}endregion ATTACHER-REMOVER
			//{region SETTER-GETTER
			public function
getStringOfArray2DMap():String {
```

```
                        return encodedString;
                }
                public function getArray2DMap():Array {
                        return arrayOf2DMap;
                }
                public function
getStringOfCustomization():String {
                        return customization;
                }
                //}endregion SETTER-GETTER

                //{region PUBLIC-ORDER
                public function
saveStringOfArray2DMap():void {
                        var myFileReference:FileReference
= new FileReference();

        myFileReference.save(getStringOfArray2DMap(),
"array2DOfMap.txt");
                }
                public static const EVENT_REMOVE:String =
"EVENT_REMOVE";
                public function remove():void {
                        removeAllExistingObjects();
                        dispatchEvent(new
Event(EVENT_REMOVE));
                }
                //}endregion PUBLIC-ORDER
        }
}
```

BIODATA PENULIS

Rizka Noviana Indriyani dilahirkan di Magetan, 14 November 1991, merupakan anak pertama dari 2 bersaudara. Penulis telah menempuh pendidikan formal yaitu MI Ma'arif Mojopurno Magetan, SMP Negeri 4 Magetan, dan SMK Telkom Sandhy Putra Malang. Pada tahun 2010, penulis melanjutkan S1 di Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Selama menempuh pendidikan S1 di Teknik Informatika ITS, penulis mengambil Rumpun Mata Kuliah Interaksi, Grafika, dan Seni (RMK IGS).

Penulis sempat aktif di berbagai organisasi dan asisten mata kuliah pada tahun-tahun awal perkuliahannya sebelum kemudian terjun ke dunia profesional sebagai *game developer* di Maulidan Productions (http://www.maulidan.com/). Penghargaan yang pernah diraih penulis selama duduk di bangku kuliah adalah Juara 1 kategori Lomba Pengembangan Permainan ITS Internal Competition 2012, Juara 1 kategori Lomba Pengembangan Permainan ITS Expo 2013, Nominator INAICTA 2013 kategori Adver*game*s dengan judul *game* Save Punyu, Nominator INAICTA 2013 kategori Adver*game*s dengan judul *game* Multishop Tycoon, Juara 2 INCREFEST 2013 dengan judul *game* Save Punyu, Juara 2 Lomba Pengembangan Permainan kategori mahasiswa Enumeration 2014, dan Best 5 Android Application Cakrawala Awards 2014. Penulis dapat dihubungi melalui surel di noviana.rizka@gmail.com.